Язык программирования MQL5: Продвинутое использование торговой платформы MetaTrader 5
Издание 2-е, исправленное и дополненное

Тимур Машнин



© Тимур Машнин, 2019

ISBN 978-5-4496-8185-0

Создано в интеллектуальной издательской системе Ridero

Исходный код

Исходный код к этой книге можно посмотреть и скачать по адресу https://github.com/novts/MetaTrader-5-Creating-Trading-Robots-and-Indicators-with-MQL5

Введение

Надеюсь, вы все уже прочитали справочник MQL5 на сайте https://www.mql5.com/ru/docs.

https://www.mql5.com/ru/docs



Здесь мы не будем пересказывать этот документ, а сосредоточимся на его практическом использовании. Мы будем лишь позволять себе изредка только его цитирование.

Как сказано в предисловии к справочнику:

Для выполнения конкретных задач по автоматизации торговых операций MQL5-программы разделены на четыре специализированных типа.

И далее идет перечисление: Советник, Пользовательский индикатор, Скрипт, Библиотека и Включаемый файл.

Скрипты используются для выполнения одноразовых действий, обрабатывая только событие своего запуска, и поэтому не будут нам здесь интересны.

Также нам не будут интересны библиотеки, так как использование включаемых файлов более предпочтительно для уменьшения накладных расходов.

Поэтому мы сосредоточимся на создании советников и индикаторов с использованием включаемых файлов. Такова наша цель применения языка программирования MQL5, синтаксис которого конечно интересен, но будет нам только в помощь.

На самом деле программирование на языке MQL5 представляет собой яркий пример событийно-ориентированного программирования, так как весь код MQL5-приложения построен на переопределении функций обратного вызова – обработчиков событий клиентского терминала и пользователя.

А уже в коде функций обратного вызова можно использовать либо процедурное программирование, либо объектно-ориентированное программирование. Здесь мы рассмотрим оба этих подхода.

Начало работы

Для начала работы выберем какого-нибудь посредника, чтобы подключиться к его серверу и получать реальные котировки рынка для разработки и тестирования наших MQL5 приложений.



Под посредником мы имеем в виду торгового представителя, юридическое лицо, профессионального участника рынка, имеющего право совершать операции на рынке по поручению клиента и за его счёт

или от своего имени и за счёт клиента на основании возмездных договоров с клиентом.

Теперь, что такое рынок?

Существуют разные типы рынков.

Это валютный рынок, это фондовый рынок или рынок ценных бумаг, это товарный рынок, и это рынок фьючерсов и опционов.

Мы с вами сосредоточимся на валютном рынке или рынке форекс.

Что такое рынок форекс?

FOREX – это сокращение от двух слов Foreign Exchange, что означает Валютный Обмен.

В отличие от других рынков, где торговля происходит на биржах, рынок форекс – это внебиржевой рынок межбанковского обмена валюты без какой-либо централизованной площадки.

Участники рынка форекс – это центральные банки, коммерческие банки, инвестиционные банки, брокеры и дилеры, пенсионные фонды, страховые компании, транснациональные корпорации и т. д.

Реально, большая часть сделок по обмену одних валют на другие происходит на ВНЕБИРЖЕВОМ рынке между крупными международными банками с использованием межбанковского информационно-торгового терминала.

И торговля идет на очень большие суммы. Минимальным лотом является сумма в 1 миллион долларов или евро, стандартным – 5 или 10 миллионов долларов.

Такая торговля валютами обеспечивает в первую очередь экспортно-импортные операции клиентов банков, и во вторую, интересы собственных торгово-инвестиционных отделов международных банков.

И совершают банки сделки как на межбанковском внебиржевом рынке, так и на валютных биржах.

Откуда берутся котировки на рынке Форекс?

Если взять, например, фондовый рынок, то там есть специальное учреждение – биржа, где торгуются определённые ценные бумаги (только там и нигде больше), и эта самая биржа и выступает единым центром распространения котировок остальным участникам, в том числе дилинговым центрам.

В случае с Форексом такого центра не существует, рынок не имеет единого места торговли и объединяет всех участников посредством современных средств передачи данных.

Поскольку основной объем торговых операций осуществляется через банковские учреждения, рынок Форекс называют международным межбанковским рынком.

Все крупнейшие участники данного рынка, международные банки, осуществляют котирование и выступают своего рода «двигателями рынка», совершая сделки либо с другими банками, либо с клиентами – инвестиционными фондами, компаниями, физическими лицами.

Все остальные участники рынка Forex запрашивают у них котировки и проводят по ним свои операции.

Выставление котировок по валютным парам международные банки производят, как правило, в электронном режиме.

И котировки формируются как на основе запросов других участников, так и в потоковом режиме (индикативном), когда банк выставляет «справочный» курс, по которому он готов совершить сделку, однако не обязан будет это делать.

Окончательная цена сделки зависит от суммы сделки, статуса участника, текущего положения на рынке и других факторов.

Индикативные и реальные котировки поступают в глобальные информационные системы (Reuters, Bloomberg, Dow Jones и др.), откуда их получают другие пользователи, в том числе и дилинговые центры.

Именно котировки, полученные от обслуживающего дилингового центра, видит трейдер в своем торговом терминале, который он использует в процессе торговли.

Таким образом, если сравнивать Форекс с биржевым рынком, то здесь отсутствует цена, единая для всех без исключения участников.

Зачастую операции совершаются по разным котировкам, причем цена будет более выгодной для второстепенных участников, имеющих налаженные контакты с основными участниками – банками, а также участников, торгующих большими объемами валюты.

В то же время, благодаря высокой ликвидности рынка котировки в большинстве случаев различаются только на 1—2 пункта, что делает практически невозможным пространственный арбитраж, когда участник покупает валюту у одного продавца по какой-либо цене, зная, что он сможет в тот же момент продать её другому покупателю на более выгодных условиях.

Теперь, что такое дилинговый центр форекс?

Дилинговый центр – это небанковская организация, обеспечивающая возможность клиентам с небольшими суммами торгового капитала на условиях маржинальной торговли заключать спекулятивные сделки.

И естественно, перед передачей котировок своим клиентам, дилинговый центр накладывает на них собственный фильтр, включающий, помимо прочего, спред, который будет составлять его заработок.

Теперь, таким образом, дилинговый центр обеспечивает возможность клиентам с небольшими суммами торгового капитала на условиях маржинальной торговли заключать спекулятивные сделки.

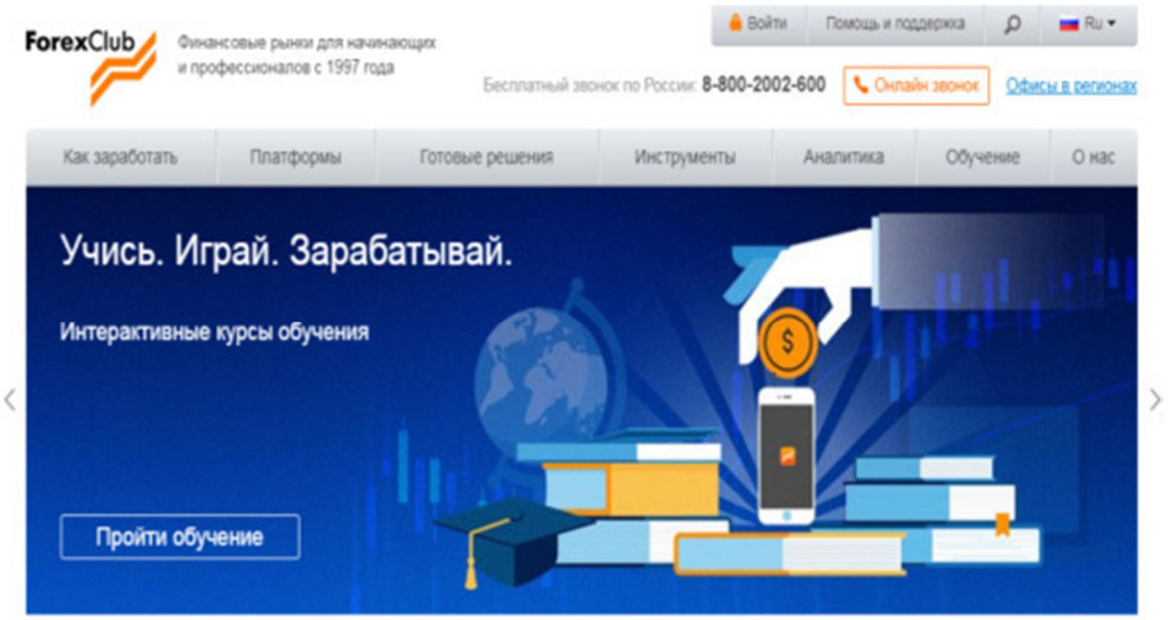
Как объясняют сами дилинговые центры, они отправляют на реальный внебиржевой рынок не все клиентские ордера, а только их агрегированную составляющую, превышающую определенный размер. А остальные ордера дилинговый центр сводит с противоположными ордерами, полученными от других клиентов.

На самом деле, как правило, ни один дилинговый центр практически никогда не выводит «сделки» своих клиентов на открытый рынок, потому как знает, что условия игры таковы, что клиент рано или поздно проиграет. Следовательно, выводить сделки на рынок нет никакой надобности.

Таким образом клиент или трейдер торгует не против рынка, а против дилингового центра.

В начале мы сказали, что нам нужен какой-нибудь посредник, чтобы подключиться к его серверу и получать реальные котировки рынка для разработки и тестирования наших MQL5 приложений.

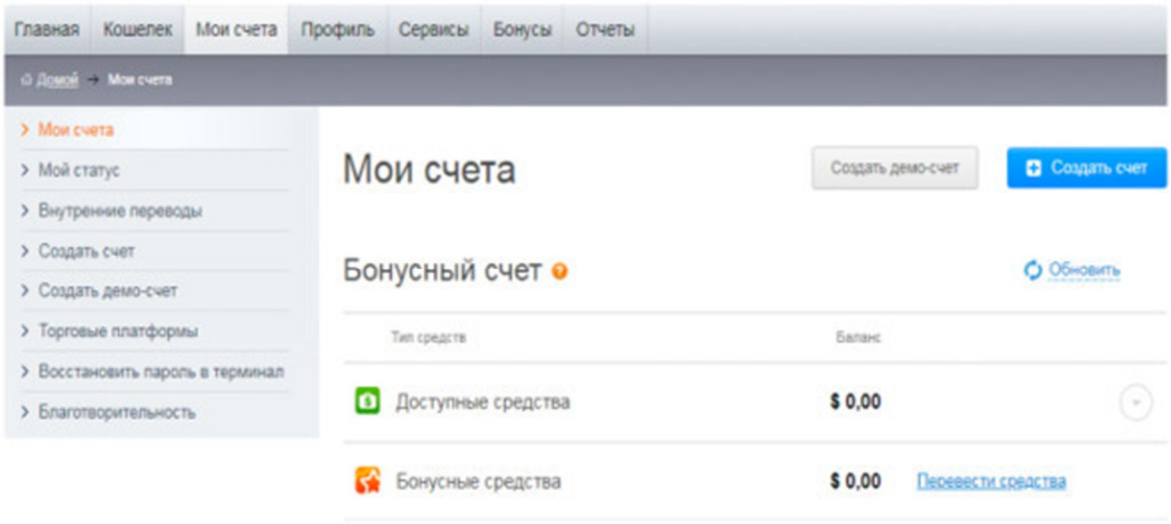
Так как у нас нет миллионов долларов, чтобы непосредственно торговать на Форексе, и мы не можем себе позволить установить свой межбанковский информационно-торговый терминал, в качестве посредника выберем дилинговый центр.



Давайте выберем, например, дилинговый центр Forex Club.

Я не являюсь фанатом данной компании, это просто для нашего кодирования.

Для реальной торговли лучше выбрать, наверное, какой-нибудь банк.



Зарегистрируемся и создадим демо-счет для платформы MetaTrader 5.

Forex Club предлагает два типа счетов:

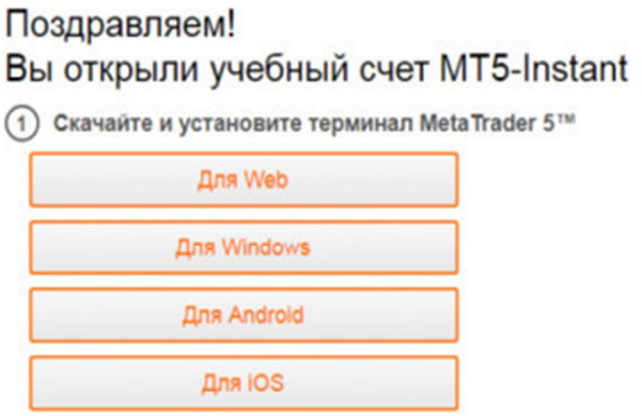
Немедленное исполнение (Instant Execution)

В этом режиме исполнение рыночного ордера осуществляется по предложенной цене. При отправке запроса на исполнение, платформа автоматически подставляет в ордер текущие цены.

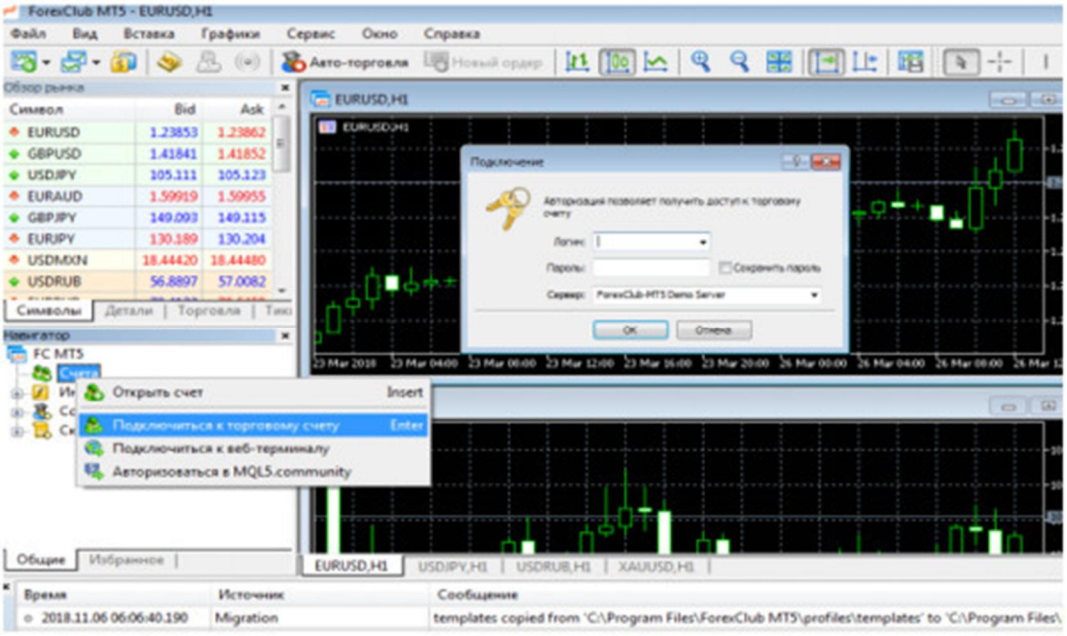
И исполнение по рынку (Market Execution)

В этом режиме исполнения рыночного ордера решение о цене исполнения принимает дилинговый центр без дополнительного согласования с трейдером.

Мы откроем счет – немедленное исполнение (Instant Execution).



Далее скачаем и установим платформу MetaTrader 5.



И подключимся к серверу Forex Club, используя логин и пароль демо счета.

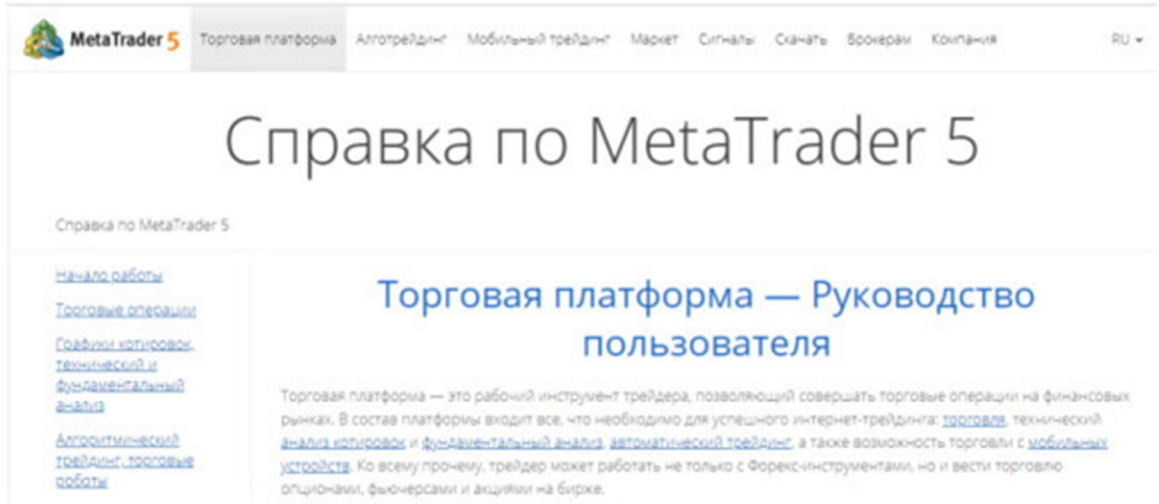
Далее, нажав правой кнопкой мышки на графике и зайдя в свойства, настроим внешний вид графика, как вам нравится.



Мультирыночная платформа MetaTrader 5 позволяет совершать торговые операции на Forex, фондовых биржах и фьючерсами.

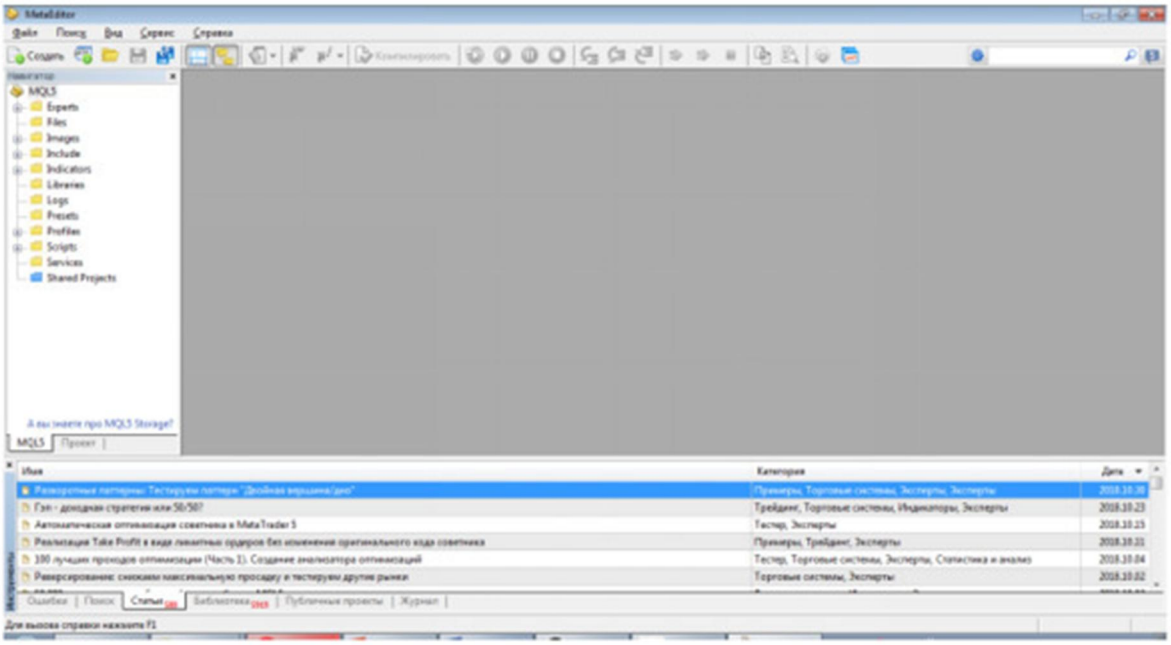
С помощью MetaTrader 5 можно также проводить технический анализ котировок, работать с торговыми роботами и копировать сделки других трейдеров.

https://www.metatrader5.com/ru/terminal/help



Более подробно про платформу MetaTrader 5 и про ее интерфейс можно почитать в соответствующей справке.

Мы не будем пересказывать эту справку, так как это было бы слишком нагло брать деньги за книгу, в которой пересказывается общедоступная справка.



Также помимо терминала MetaTrader 5, нас интересует редактор MQL5, который можно открыть либо с помощью ярлыка, либо в меню Сервис терминала MetaTrader 5.

MetaEditor – это современная среда разработки торговых стратегий, интегрированная с платформой MetaTrader.

С помощью MetaEditor можно создавать торговых роботов, технические индикаторы, скрипты, графические панели управления и многое другое.

https://www.metatrader5.com/ru/metaeditor/help

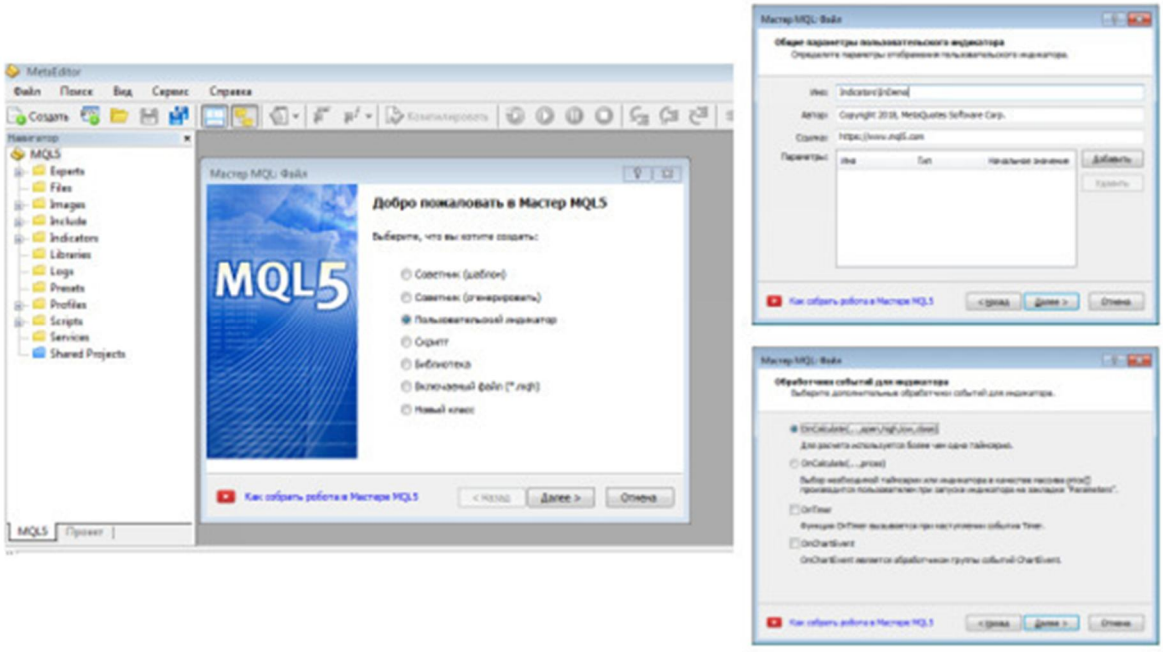


Для редактора MetaEditor также есть подробная справка, которую мы также не будем пересказывать.

Мы лучше сразу займемся практическим кодированием.

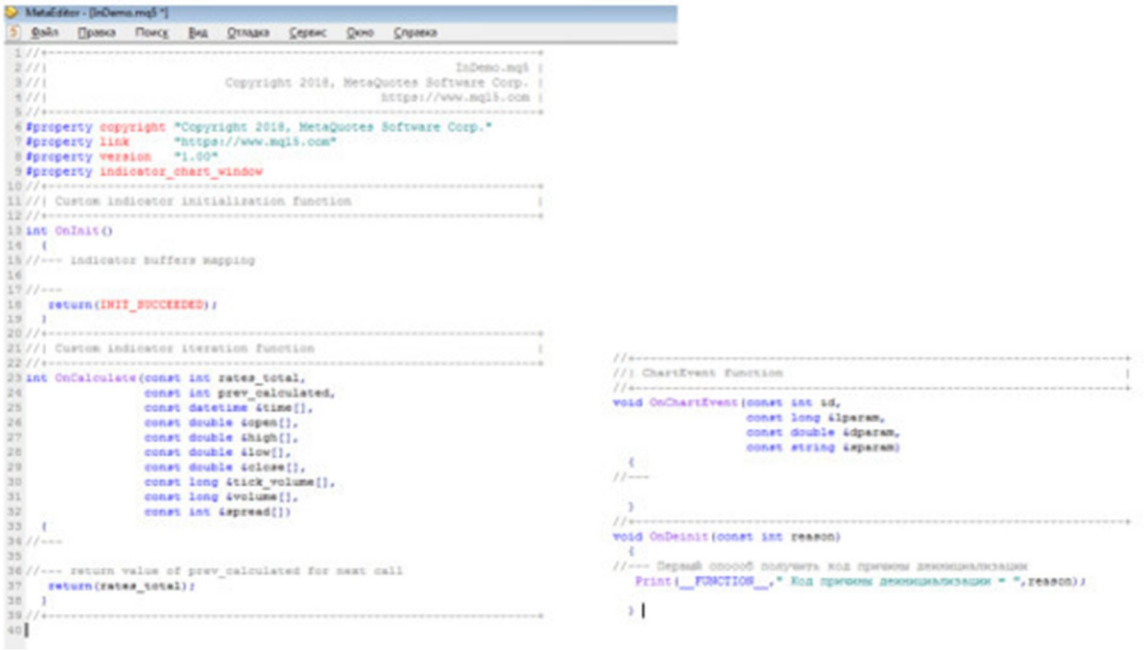
Общая структура индикатора

Для создания основы пользовательского индикатора используем редактор MetaEditor.



Нажмем кнопку меню Создать и в окне мастера выберем Пользовательский индикатор.

Нажмем Далее, введем имя создаваемого индикатора, нажмем Далее и отметим функции, которые мастер должен сгенерировать и в следующем окне нажмем Готово.



В результате будет создан код основы индикатора.

Код индикатора начинается с блока объявления свойств индикатора и различных объектов, используемых индикатором, таких как массивы буферов индикатора, параметры ввода, глобальные переменные, хэндлы используемых технических индикаторов, константы.

Данный блок кода выполняется приложением Торговая Платформа MetaTrader 5 сразу при присоединении индикатора к графику символа.

После блока объявления свойств индикатора, его параметров и переменных, идет описание функций обратного вызова, которые терминал вызывает при наступлении таких событий, как инициализация индикатора после его загрузки, перед деинициализацией индикатора, при изменении ценовых данных, при изменении графика символа пользователем.

Для обработки вышеуказанных событий необходимо описать такие функции как OnInit (), OnDeinit (), OnCalculate () и OnChartEvent ().

В функции OnInit () индикатора, как правило, объявленные в начальном блоке массивы связываются с буферами индикатора, определяя его

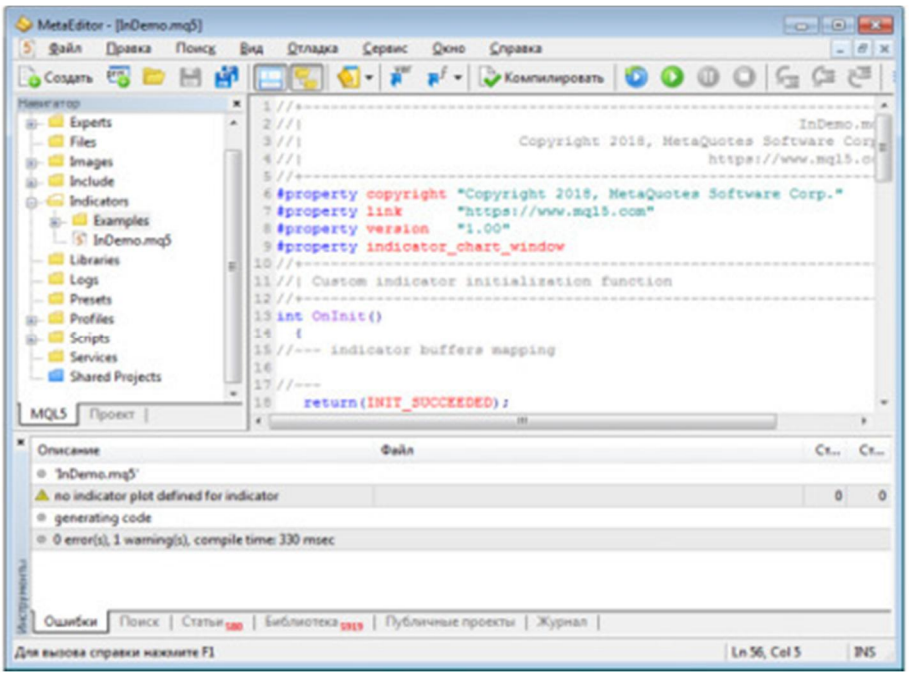
выводимые значения, задаются цвета индикатора, точность отображения значений индикатора, его подписи и другие параметры отображения индикатора. Кроме того, в функции OnInit () индикатора могут получаться хэндлы используемых технических индикаторов и рассчитываться другие используемые переменные.

В функции OnDeinit () индикатора, как правило, с графика символа удаляются графические объекты индикатора, а также удаляются хэндлы используемых технических индикаторов.

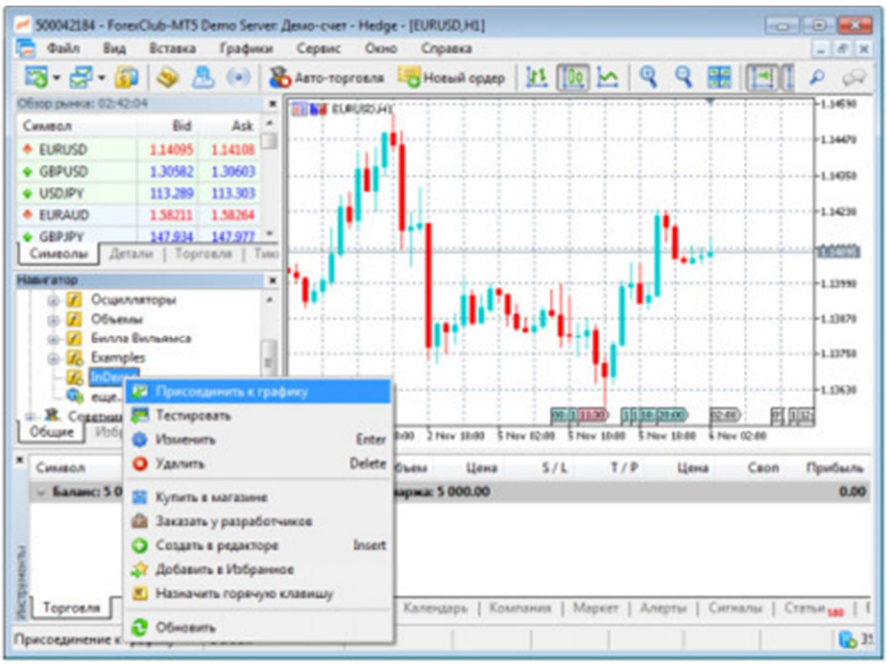
В функции OnCalculate () собственно и производится расчет значений индикатора, заполняя ими объявленные в начальном блоке массивы, которые в функции OnInit () индикатора были связаны с буферами индикатора, данные из которых берутся терминалом для отрисовки индикатора. Кроме того, в функции OnCalculate () могут изменяться цвета индикатора и другие параметры его отображения.

В функции OnChartEvent () могут обрабатываться события, генерируемые другими индикаторами на графике, а также удаление пользователем графического объекта индикатора и другие события, возникающие при работе пользователя с графиком.

На этом код индикатора заканчивается, хотя там могут быть также определены пользовательские функции, которые вызываются из функций обратного вызова OnInit (), OnDeinit (), OnCalculate () и OnChartEvent ().



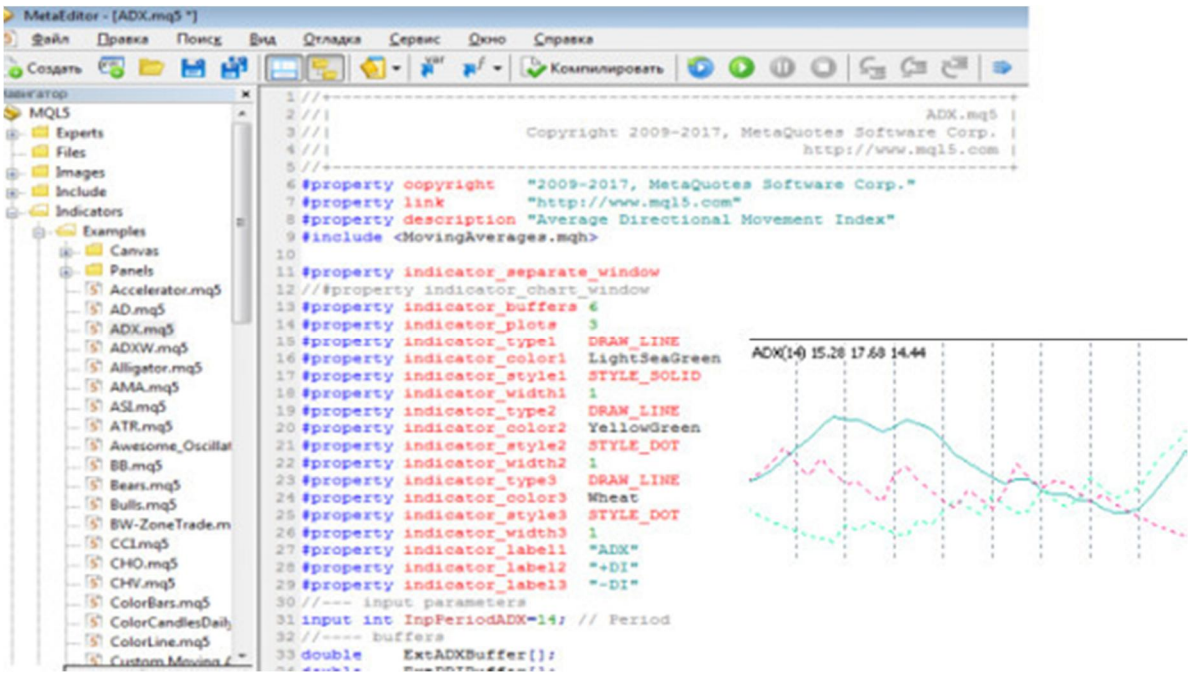
Для компиляции нашего индикатора нажмем кнопку Компилировать редактора, при этом в нижнем окне отобразится результат компиляции.



После компиляции наш индикатор автоматически появится в торговом терминале и мы сможем присоединить его к графику финансового инструмента.

Свойства индикатора

Давайте более подробно рассмотрим свойства индикатора.



Цитата из справочника:

Свойства программ (#property). У каждой mql5-программы можно указать дополнительные специфические параметры #property, которые помогают клиентскому терминалу правильно обслуживать программы без необходимости их явного запуска. В первую очередь это касается внешних настроек индикаторов. Свойства, описанные во включаемых файлах, полностью игнорируются. Свойства необходимо задавать в главном mq5-файле: #property идентификатор значение.

Включаемый файл указывается с помощью ключевого слова #include, после которого следует путь к включаемому файлу.

Включаемый файл – это часто используемый блок кода. Такие файлы могут включаться в исходные тексты экспертов, скриптов, пользовательских индикаторов и библиотек на этапе компиляции. Использование включаемых файлов более предпочтительно, чем использование библиотек, из-за дополнительных накладных расходов при вызове библиотечных функций.

Включаемые файлы могут находиться в той же директории, что и исходный файл, в этом случае используется директива #include с двойными кавычками. Другое место хранения включаемых файлов – в директории <каталог_терминала> \MQL5\Include, в этом случае используется директива #include с угловыми скобками.

В качестве первого свойства индикатора, как правило, указывается имя разработчика, например:

#property copyright

Далее указывается ссылка на сайт разработчика:

#property link

После этого идет описание индикатора, каждая строка которого обозначается с помощью идентификатора description, например:

#property description «Average Directional Movement Index»

Далее указывается версия индикатора:

#property version «1.00»

На этом, как правило, объявление общих свойств индикатора заканчивается.

Индикатор может появляться в окне терминала двумя способами – на графике символа или в отдельном окне под графиком символа.

Свойство:

#property indicator_chart_window

Определяет отрисовку индикатора на графике символа.

А свойство:

#property indicator_separate_window

Определяет вывод индикатора в отдельное окно.

Одно из самых важных свойств индикатора – это количество буферов для расчета индикатора, например:

#property indicator_buffers 6

Данное свойство тесно связано с двумя другими свойствами индикатора – количеством графических построений и видом графических построений.

Количество графических построений это количество цветных диаграмм, составляющих индикатор.

Например, для индикатора ADX:

#property indicator_plots 3

Индикатор состоит из трех диаграмм (линий) – индикатора направленности +DI, индикатора направленности —DI и самого индикатора ADX.

Вид графических построений – это та графическая форма, из которой составляется график индикатора.

Например, для индикатора ADX:

#property indicator_type1 DRAW_LINE

#property indicator_type2 DRAW_LINE

#property indicator_type3 DRAW_LINE

Таким образом, каждая диаграмма индикатора ADX – это линия.

Графическая форма сопоставляется с графическим построением с помощью номера графического построения, следующего после indicator_type.

Цвет каждого графического построения индикатора задается свойством indicator_colorN.

Например, для индикатора ADX:

#property indicator_color1 LightSeaGreen

#property indicator_color2 YellowGreen

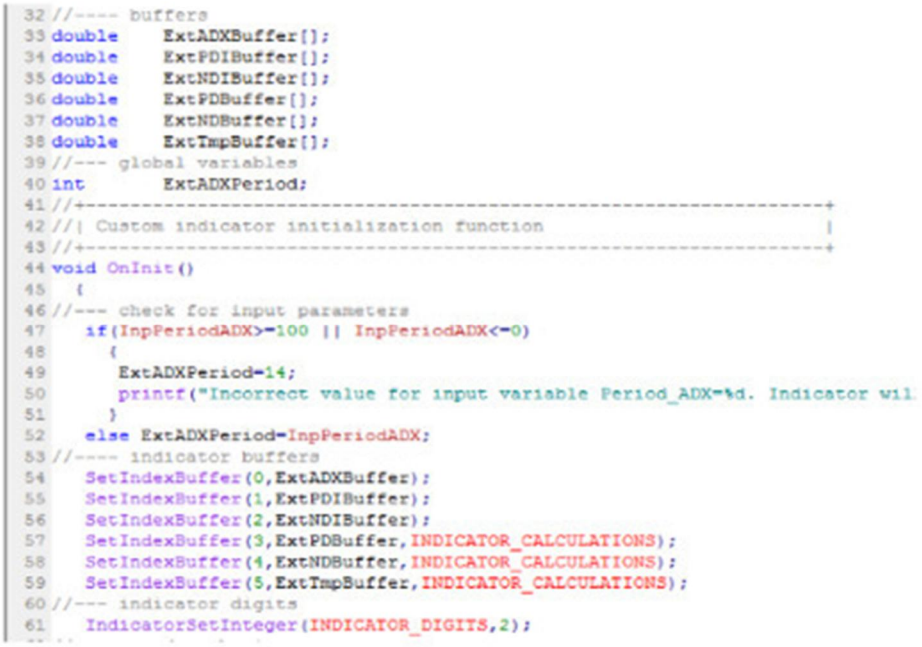
#property indicator_color3 Wheat

Цвет сопоставляется с графическим построением с помощью номера графического построения, следующего после indicator_color.

https://www.mql5.com/ru/docs/constants/objectconstants/webcolors



В справочнике MQL5 есть таблица Web-цветов для определения цвета графического построения.



```
//---- buffers
double    ExtADXBuffer[];
double    ExtPDIBuffer[];
double    ExtNDIBuffer[];
double    ExtPDBuffer[];
double    ExtNDBuffer[];
double    ExtTmpBuffer[];
//--- global variables
int       ExtADXPeriod;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- check for input parameters
   if(InpPeriodADX>=100 || InpPeriodADX<=0)
     {
      ExtADXPeriod=14;
      printf("Incorrect value for input variable Period_ADX=%d. Indicator wil
     }
   else ExtADXPeriod=InpPeriodADX;
//---- indicator buffers
   SetIndexBuffer(0,ExtADXBuffer);
   SetIndexBuffer(1,ExtPDIBuffer);
   SetIndexBuffer(2,ExtNDIBuffer);
   SetIndexBuffer(3,ExtPDBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(4,ExtNDBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(5,ExtTmpBuffer,INDICATOR_CALCULATIONS);
//--- indicator digits
   IndicatorSetInteger(INDICATOR_DIGITS,2);
```

Вернемся теперь к количеству буферов для расчета индикатора.

Так как данные для построения каждой диаграммы индикатора берутся из своего буфера индикатора, количество заявленных буферов индикатора не может быть меньше, чем заявленное число графических построений индикатора.

Сразу же возникает вопрос, каким образом конкретный массив, представляющий буфер индикатора, сопоставляется с конкретным графическим построением индикатора.

Делается это в функции обратного вызова OnInit () с помощью вызова функции SetIndexBuffer.

Например, для индикатора ADX:

SetIndexBuffer (0,ExtADXBuffer);

SetIndexBuffer (1,ExtPDIBuffer);

SetIndexBuffer (2,ExtNDIBuffer);

Где первый аргумент, это номер графического построения.

Таким образом, массив связывается с диаграммой индикатора, а диаграмма связывается с ее формой и цветом.

Однако с буферами индикатора все немного сложнее.

Их количество может быть заявлено больше, чем количество графических построений индикатора.

Что это означает?

Это означает, что некоторые массивы, представляющие буфера индикатора, используются не для построения диаграмм индикатора, а для промежуточных вычислений.

Например, для индикатора ADX:

SetIndexBuffer (3,ExtPDBuffer, INDICATOR_CALCULATIONS);

SetIndexBuffer (4,ExtNDBuffer, INDICATOR_CALCULATIONS);

SetIndexBuffer (5,ExtTmpBuffer, INDICATOR_CALCULATIONS);

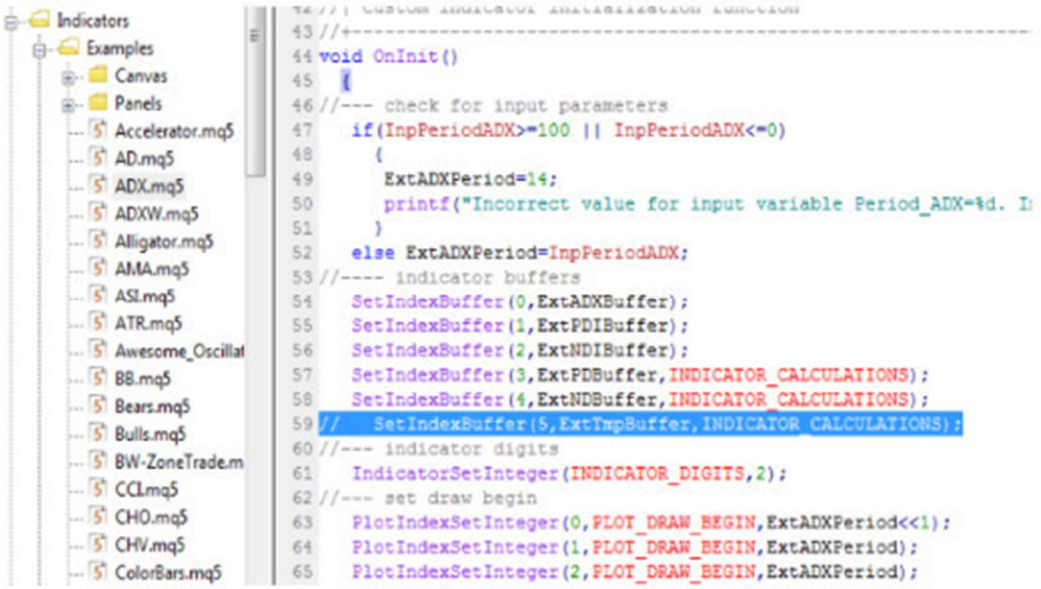
Такой массив определяется с помощью третьего параметра INDICATOR_CALCULATIONS.

Это дает следующее:

Все дело в частичном заполнении массива.

Если массив, указанный в функции SetIndexBuffer, является динамическим, т.е. объявлен без указания размера, но он привязан к буферу индикатора с помощью функции SetIndexBuffer, клиентский терминал сам заботится о том, чтобы размер такого массива соответствовал ценовой истории.

Рассмотрим это на примере индикатора ADX.



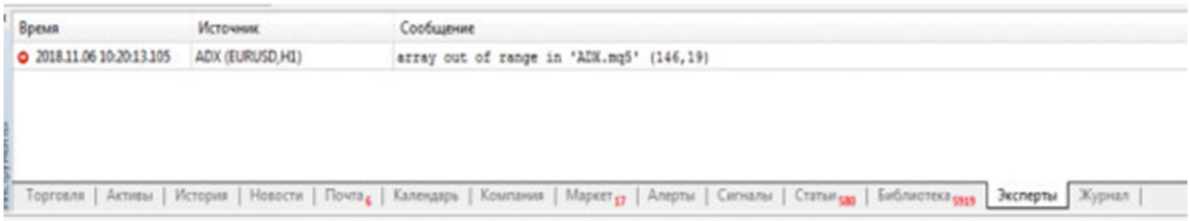
В редакторе MQL5, в окне Navigator (Навигатор), в разделе Indicators-> Examples выберем и откроем исходный код индикатора ADX.

В функции OnInit () закомментируем строку:

// SetIndexBuffer (5,ExtTmpBuffer, INDICATOR_CALCULATIONS);

Теперь массив ExtTmpBuffer является просто динамическим массивом.

Откомпилируем код индикатора и присоединим индикатор к графику в терминале MetaTrader 5.



В результате Терминал выдаст ошибку.

array out of range

Это произошло потому, что мы перед заполнением данного массива значениями не указали его размера и не зарезервировали под него память.

Так что его размер был равен нулю, когда мы попытались в него что-то записать.

Статическим мы этот массив сделать тоже не можем, т.е. объявить его сразу с указанием размера, так как значения такого промежуточного массива рассчитываются в функции обратного вызова OnCalculate на основе загруженной в функцию OnCalculate истории цен, а именно массивов open [], high [], low [], и close [].

Но точный размер массивов open [], high [], low [], и close [] неизвестен, он обозначается лишь переменной rates_total.

```
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
ArrayResize(ExtTmpBuffer,rates_total);
//--- checking for bars count
   if(rates_total<ExtADXPeriod)
      return(0);
//--- detect start position
   int start;
   if(prev_calculated>1) start=prev_calculated-1;
   else
     {
      start=1;
      ExtPDIBuffer[0]=0.0;
      ExtNDIBuffer[0]=0.0;
      ExtADXBuffer[0]=0.0;
     }
```

Хорошо, но мы можем в функции OnCalculate применить функцию ArrayResize, чтобы установить размер массива:

ArrayResize (ExtTmpBuffer, rates_total);

Передав в функцию в качестве аргумента переменную rates_total – количество баров на графике, на котором запущен индикатор.

Теперь после компиляции индикатор заработает как надо.

Но дело в том, что в функции OnCalculate мы сначала рассчитываем индикатор для всей ценовой истории, т.е. для rates_total значений, а затем при поступлении нового тика по символу индикатора, и соответственно вызове функции OnCalculate, мы рассчитываем значение индикатора для этого нового тика по символу и записываем новое значение индикатора в его массив буфера.

Чтобы это реализовать с промежуточным массивом, нужно внимательно следить за его размером и записывать новое значение в конец массива.

Вместо всего этого, проще всего привязать промежуточный массив к буферу индикатора с помощью функции SetIndexBuffer и таким образом решить все эти проблемы.



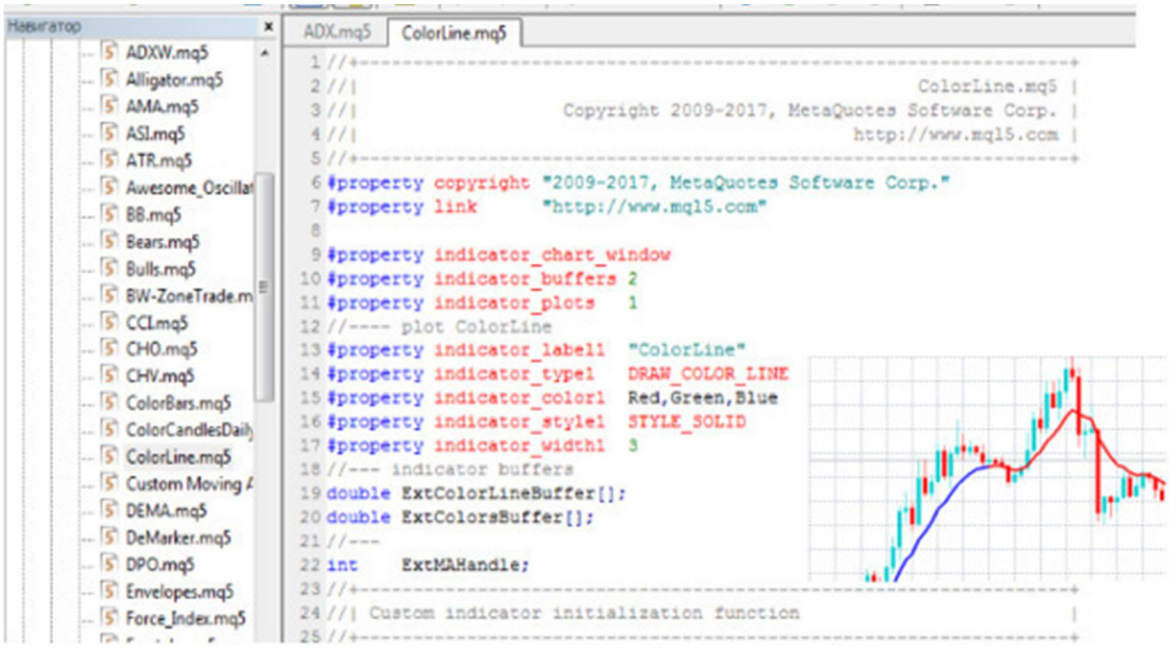

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   int       start_pos,            // откуда начнем
   int       count,                // сколько копируем
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

Аналогичная ситуация возникает, когда значения таких промежуточных массивов заполняются с помощью функции CopyBuffer, когда мы строим пользовательский индикатор на основе других индикаторов.

Функция CopyBuffer распределяет размер принимающего массива под размер копируемых данных.

Если копируется вся ценовая история, то проблем нет и в этом случае использовать INDICATOR_CALCULATIONS необязательно.

Если же мы хотим скопировать только одно новое поступившее значение, функция CopyBuffer определит размер принимающего массива как 1, и нужно будет использовать этот принимающий массив как еще один массив-посредник, из которого уже записывать значение в промежуточный массив индикатора. И в этом случае просто функцией ArrayResize для принимающего массива проблему не решить.



Теперь что нам делать, если мы хотим раскрашивать наши диаграммы индикатора в разные цвета в зависимости от цены?

Во-первых, мы должны указать, что наша графическая форма нашего графического построения является цветной, например:

#property indicator_type1 DRAW_COLOR_LINE

В идентификатор геометрической формы добавляется слово COLOR.

Далее значение свойства #property indicator_buffers увеличивается на единицу и объявляется еще один массив для хранения цвета.

```
#property indicator_buffers 2
#property indicator_plots   1
//---- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  Red,Green,Blue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- indicator buffers
double ExtColorLineBuffer[];
double ExtColorsBuffer[];
//---
int    ExtMAHandle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ExtColorsBuffer,INDICATOR_COLOR_INDEX);
//--- get MA handle
   ExtMAHandle=iMA(Symbol(),0,10,0,MODE_EMA,PRICE_CLOSE);
  }
```

Функцией SetIndexBuffer объявленный дополнительный массив сопоставляется с буфером цвета индикатора, например:

SetIndexBuffer (1,ExtColorsBuffer, INDICATOR_COLOR_INDEX);

В свойстве #property indicator_color, раскрашиваемого графического построения, указывается несколько цветов, например:

#property indicator_color1 Red, Green, Blue

```
//+------------------------------------------------------------------+
int getIndexOfColor(int i)
  {
   int j=i%300;
   if(j<100) return(0);// first index
   if(j<200) return(1);// second index
   return(2); // third index
  }
```

```
      //--- now set line color for every bar
      for(int i=0;i<rates_total && !IsStopped();i++)
         ExtColorsBuffer[i]=getIndexOfColor(i);
   }
```

И, наконец, каждому элементу массива, представляющего буфер цвета индикатора, присваивается номер цвета, определенный в свойстве #property indicator_color.

В данном случае, это 0, 1 и 2.

Теперь при отрисовке диаграммы индикатора, из буфера берется значение диаграммы, по позиции значения оно сопоставляется со значением буфера цвета, и элемент диаграммы становится цветным.

```
//---- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
//#property indicator_color1  Red,Green,Blue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- indicator buffers
double ExtColorLineBuffer[];
double ExtColorsBuffer[];
//---
int    ExtMAHandle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {

  PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,3);
  PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,Red);
  PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,Green);
  PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,Blue);
```

Вместо свойства #property indicator_color, цвета графического построения можно задать программным способом:

Задаем количество индексов цветов для графического построения с помощью функции:

PlotIndexSetInteger (0,PLOT_COLOR_INDEXES,3);

И задаем цвет для каждого индекса с помощью функции:

PlotIndexSetInteger (0,PLOT_LINE_COLOR,0,Red);

Где первый параметр – индекс графического построения, соответственно первое графическое построение имеет индекс 0.

Это идентично объявлению:

#property indicator_color1 Red, Green, Blue

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Average Directional Movement Index"
#include <MovingAverages.mqh>

#property indicator_separate_window
//#property indicator_chart_window
#property indicator_buffers 6
#property indicator_plots   3
#property indicator_type1   DRAW_LINE
#property indicator_color1  LightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"
```

Давайте продолжим рассмотрение свойств индикатора.

Толщина линии диаграммы индикатора задается свойством indicator_widthN, где N – номер графического построения, например:

#property indicator_width1 1

Также можно задать стиль линии диаграммы индикатора – сплошная линия, прерывистая, пунктирная, штрих-пунктирная, штрих – с помощью свойства indicator_styleN, где N – номер графического построения, например:

#property indicator_style1 STYLE_SOLID

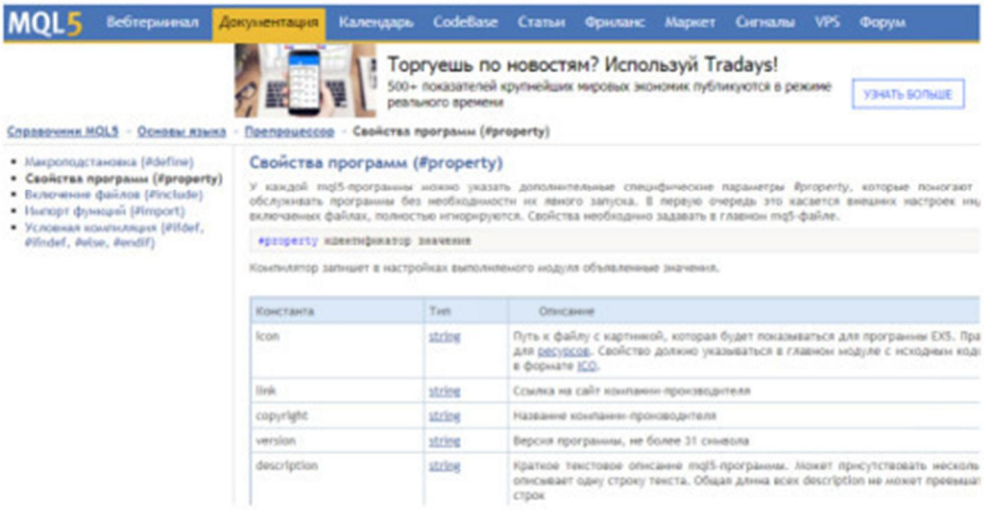
И, наконец, свойство indicator_labelN указывает метки диаграмм индикатора в DataWindow или Окно данных, например:

#property indicator_label1 «ADX»

#property indicator_label2 "+DI»

#property indicator_label3 "-DI»

https://www.mql5.com/ru/docs/basis/preprosessor/compilation

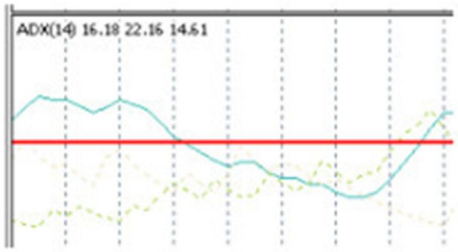


Другие свойства индикатора можно посмотреть в справочнике.

Правда можно отметить еще одну группу свойств, которая позволяет нарисовать горизонтальный уровень индикатора в отдельном окне, например:

```
#property indicator_separate_window
//#property indicator_chart_window
#property indicator_buffers 6
#property indicator_plots   3
#property indicator_type1   DRAW_LINE
#property indicator_color1  LightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"

#property indicator_level1 30.0
#property indicator_levelcolor Red
#property indicator_levelstyle STYLE_SOLID
#property indicator_levelwidth 2
```



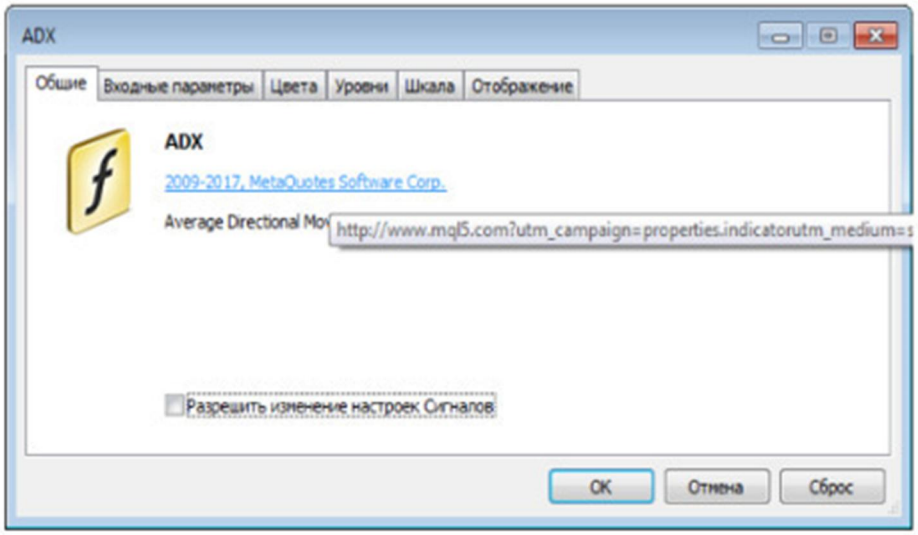
#property indicator_level1 30.0

#property indicator_levelcolor Red

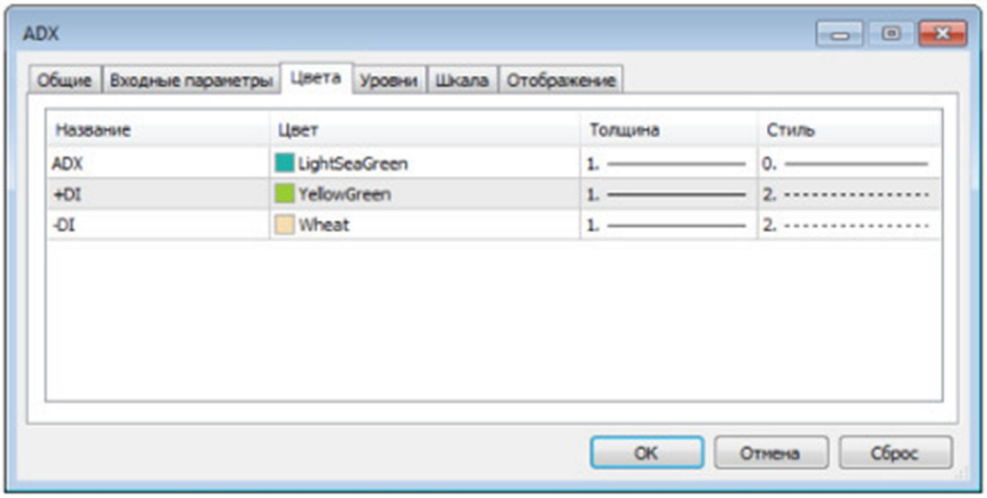
#property indicator_levelstyle STYLE_SOLID

#property indicator_levelwidth 2

В результате добавления этих строк кода в индикатор ADX, у него появится горизонтальный уровень.



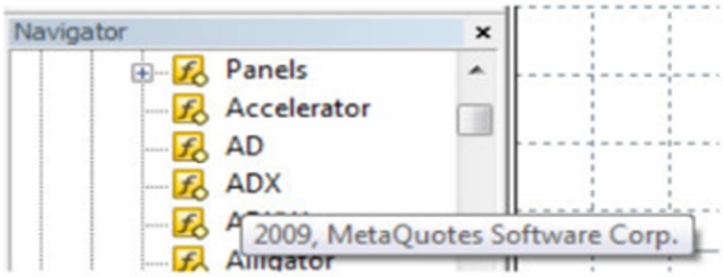
Теперь, на примере индикатора ADX, при присоединении индикатора к графику в MetaTrader 5, во-первых, откроется диалоговое окно индикатора, которое во вкладке Общие отобразит значения свойств copyright, link и description.



А во вкладке Цвета отобразятся значения свойств indicator_label, indicator_color, indicator_width, indicator_style.

Само же название индикатора определяется именем файла индикатора.

К слову сказать, диалоговое окно индикатора можно открыть и после присоединения индикатора к графику, с помощью контекстного меню, щелкнув правой кнопкой мышки на индикаторе и выбрав свойства индикатора.



При наведении курсора на название индикатора в окне Navigator терминала всплывает подсказка, отображающая свойство copyright.




```
#property indicator_label1  "ADX"
```

```
67 //--- indicator short name
68    string short_name="ADX("+string(ExtADXPeriod)+")";
69    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
```

После присоединения индикатора свойство:

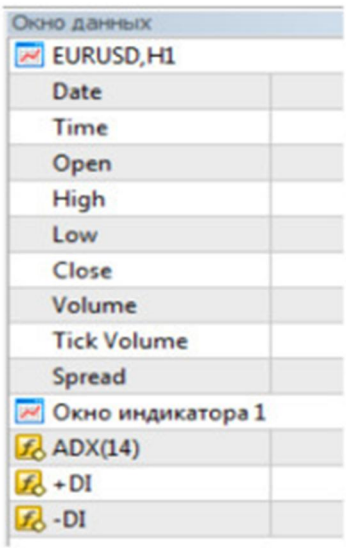
#property indicator_label1 «ADX»

работать не будет, так как в функции OnInit () с помощью вызова функции:

string short_name=«ADX (»+string (ExtADXPeriod) +»)»;

IndicatorSetString (INDICATOR_SHORTNAME, short_name);

изменена подпись индикатора на ADX (14) – период индикатора.




```
70 //--- change 1-st index label
71    PlotIndexSetString(0,PLOT_LABEL,short_name);
```

А вызовом функции:

PlotIndexSetString (0,PLOT_LABEL, short_name);

изменена метка индикатора в окне Окно Данных, которое открывается в меню Вид терминала.

Значения же свойств:

#property indicator_label2 "+DI»

#property indicator_label3 "-DI»

отображаются, как и было определено, во всплывающих подсказках к диаграммам индикатора и отображаются в окне Окно Данных.

```
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Average Directional Movement Index"
#include <MovingAverages.mqh>

#property indicator_separate_window
//#property indicator_chart_window
#property indicator_buffers 6
#property indicator_plots   3
#property indicator_type1   DRAW_LINE
#property indicator_color1  LightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"
```

```
//---- indicator buffers
   SetIndexBuffer(0,ExtADXBuffer);
   SetIndexBuffer(1,ExtPDIBuffer);
   SetIndexBuffer(2,ExtNDIBuffer);
   SetIndexBuffer(3,ExtPDBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(4,ExtNDBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(5,ExtTmpBuffer,INDICATOR_CALCULATIONS);
```

В коде индикатора ADX объявленное количество буферов индикатора больше, чем количество графических построений.

Свойство indicator_buffers равно 6

А свойство indicator_plots равно 3

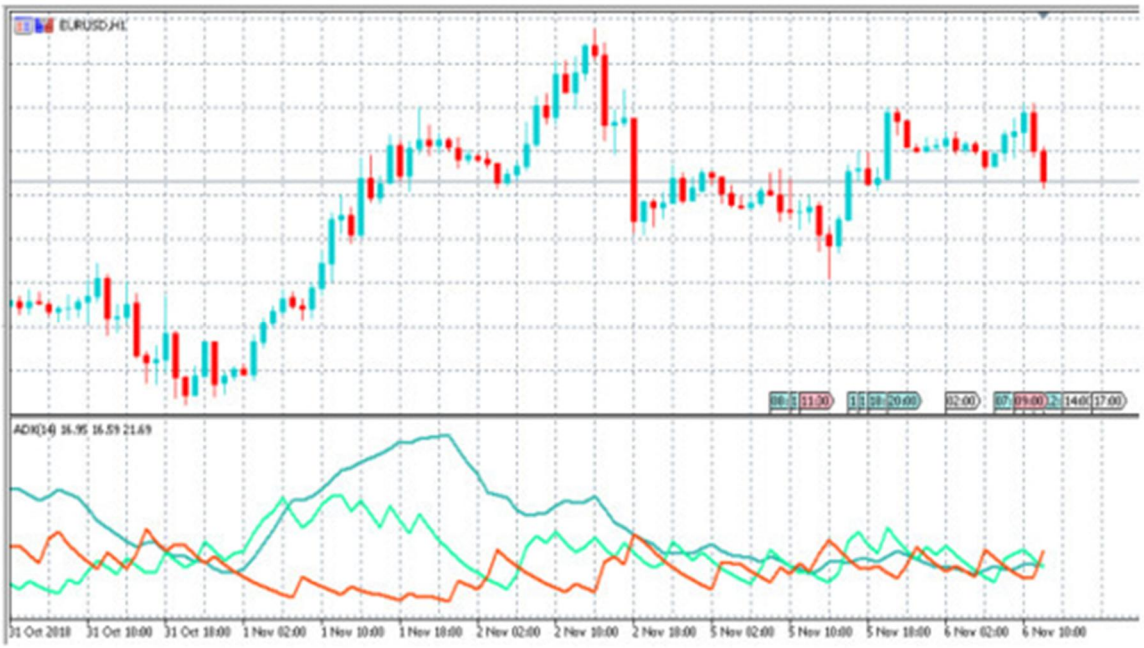
Сделано это для того, чтобы использовать три буфера индикатора для промежуточных расчетов.

Это массивы ExtPDBuffer, ExtNDBuffer и ExtTmpBuffer.

В функции OnCalculate индикатора, значения массивов ExtPDBuffer, ExtNDBuffer, ExtTmpBuffer рассчитываются на основе загруженной ценовой истории, а затем уже на их основе рассчитываются значения

массивов ExtADXBuffer, ExtPDIBuffer, ExtNDIBuffer, которые используются для отрисовки диаграмм индикатора.

Как уже было сказано, буферы индикатора для промежуточных вычислений здесь объявляются с константой INDICATOR_CALCULATIONS, так как заранее неизвестен размер загружаемой ценовой истории.



Теперь, в описании индикатора ADX сказано, что:

Сигнал на покупку формируется тогда, когда +DI поднимается выше – DI и при этом сам ADX растет.

В момент, когда +DI расположен выше – DI, но сам ADX начинает снижаться, индикатор подает сигнал о том, что рынок «перегрет» и пришло время фиксировать прибыль.

Сигнал на продажу формируется тогда, когда +DI опускается ниже – DI и при этом ADX растет.

В момент, когда +DI расположен ниже – DI, но сам ADX начинает снижаться, индикатор подает сигнал о том, что рынок «перегрет» и пришло время фиксировать прибыль.

Давайте, модифицируем код индикатора ADX таким образом, чтобы раскрасить диаграмму ADX в четыре цвета, которые соответствуют описанным выше четырем торговым сигналам.

```
//#property indicator_chart_window
#property indicator_buffers 7
#property indicator_plots   3
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  LightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"

//--- input parameters
input int InpPeriodADX=14; // Period
//---- buffers
double    ExtADXBuffer[];
double    ExtPDIBuffer[];
double    ExtNDIBuffer[];
double    ExtPDBuffer[];
double    ExtNDBuffer[];
double    ExtTmpBuffer[];
//Буфер цвета
double    ExtColorsBuffer[];
```

В качестве первого шага изменим свойство indicator_type1 на DRAW_COLOR_LINE.

Далее увеличим на единицу значение свойства indicator_buffers на значение 7.

Объявим массив для буфера цвета ExtColorsBuffer.

```
void OnInit()
  {
//--- check for input parameters
   if(InpPeriodADX>=100 || InpPeriodADX<=0)
     {
      ExtADXPeriod=14;
      printf("Incorrect value for input variable Period_ADX=%d.
     }
   else ExtADXPeriod=InpPeriodADX;
//---- indicator buffers
SetIndexBuffer(0,ExtADXBuffer);
SetIndexBuffer(1,ExtColorsBuffer,INDICATOR_COLOR_INDEX);
SetIndexBuffer(2,ExtPDIBuffer);
SetIndexBuffer(3,ExtNDIBuffer);
SetIndexBuffer(4,ExtPDBuffer,INDICATOR_CALCULATIONS);
SetIndexBuffer(5,ExtNDBuffer,INDICATOR_CALCULATIONS);
SetIndexBuffer(6,ExtTmpBuffer,INDICATOR_CALCULATIONS);
```

И в функции OnInit () свяжем объявленный массив с буфером цвета с помощью функции SetIndexBuffer.

Тут есть хитрость – индекс буфера цвета должен следовать за индексом буфера значений индикатора.

Если, например, связать массив ExtColorsBuffer с буфером с индексом 6, тогда индикатор не будет корректно отрисовываться.

```
#property indicator_separate_window
//#property indicator_chart_window
#property indicator_buffers 7
#property indicator_plots   3
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  LightSeaGreen, clrBlue, clrLightBlue, clrRed, clrLightPink
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_type2   DRAW_LINE
#property indicator_color2  YellowGreen
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
#property indicator_type3   DRAW_LINE
#property indicator_color3  Wheat
#property indicator_style3  STYLE_DOT
#property indicator_width3  1
#property indicator_label1  "ADX"
#property indicator_label2  "+DI"
#property indicator_label3  "-DI"
```

В свойство indicator_color1 добавим цветов.

И увеличим толщину линии с помощью свойства indicator_width1.

```
ExtColorsBuffer[i]=0;
if(ExtPDIBuffer[i]>ExtNDIBuffer[i]&&ExtADXBuffer[i]>ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=1;
}
if(ExtPDIBuffer[i]>ExtNDIBuffer[i]&&ExtADXBuffer[i]<ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=2;
}
if(ExtPDIBuffer[i]<ExtNDIBuffer[i]&&ExtADXBuffer[i]>ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=3;
}
if(ExtPDIBuffer[i]<ExtNDIBuffer[i]&&ExtADXBuffer[i]<ExtADXBuffer[i-1]){
ExtColorsBuffer[i]=4;
}
```

В функции OnCalculate в конце перед закрывающей скобкой цикла for добавим код заполнения буфера цвета значениями согласно описанной нами стратегии.



Откомпилируем код и получим индикатор с визуальным отображением сигналов на покупку и продажу:

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Relative Strength Index"
//--- indicator settings
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
#property indicator_level1 30
#property indicator_level2 70
#property indicator_buffers 3
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  DodgerBlue
//--- input parameters
input int InpPeriodRSI=14; // Period
//--- indicator buffers
double    ExtRSIBuffer[];
double    ExtPosBuffer[];
double    ExtNegBuffer[];
//--- global variable
int       ExtPeriodRSI;
//+------------------------------------------------------------------+
```

В редакторе MQL5 откроем другой индикатор из папки Examples – RSI.

Данный индикатор имеет два ключевых уровня, которые определяют области перекупленности и перепроданности.

В коде индикатора эти уровни определены как свойства:

#property indicator_level1 30

#property indicator_level2 70

Давайте улучшим отображение этих уровней, добавив им цвета и стиля.

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Relative Strength Index"
//--- indicator settings
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
#property indicator_level1 30
#property indicator_level2 70
#property indicator_buffers 3
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  DodgerBlue

#property indicator_levelcolor Red
#property indicator_levelstyle STYLE_SOLID
#property indicator_levelwidth 1
```

Для этого добавим свойства:

#property indicator_levelcolor Red

#property indicator_levelstyle STYLE_SOLID

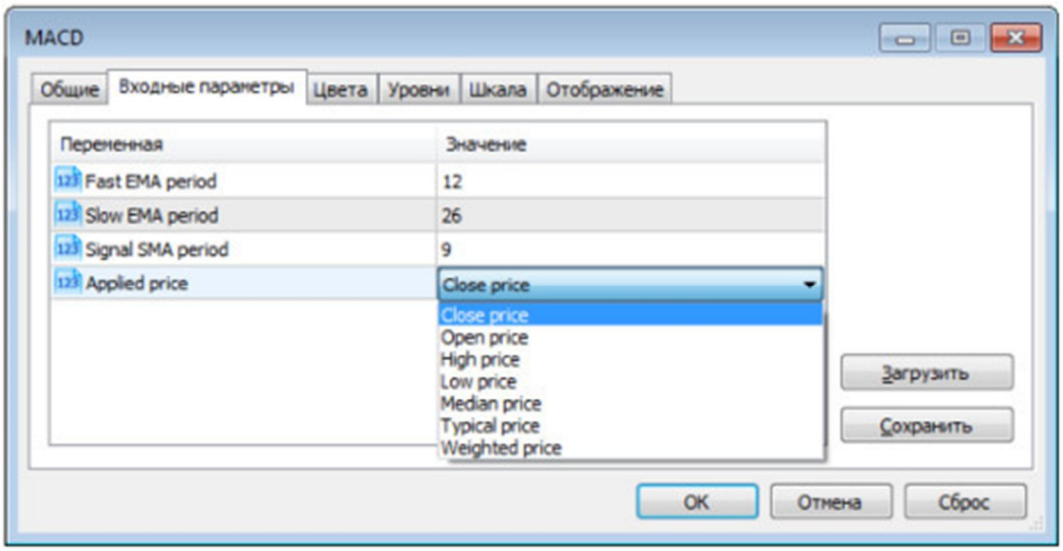
#property indicator_levelwidth 1

Теперь индикатор будет выглядеть следующим образом.



Параметры ввода и переменные индикатора

Параметры ввода это те параметры индикатора, которые отображаются пользователю перед присоединением индикатора к графику во вкладке Входные параметры диалогового окна.



Например, для индикатора MACD – это периоды скользящих средних и тип применяемой цены.

Здесь пользователь может поменять параметры индикатора по умолчанию, и индикатор присоединится к графику с уже измененными параметрами.

Также пользователь может поменять параметры индикатора после присоединения индикатора к графику, щелкнув правой кнопкой мышки на индикаторе и выбрав свойства индикатора.

```
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Moving Average Convergence/Divergence"
#include <MovingAverages.mqh>
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_type2   DRAW_LINE
#property indicator_color1  Silver
#property indicator_color2  Red
#property indicator_width1  2
#property indicator_width2  1
#property indicator_label1  "MACD"
#property indicator_label2  "Signal"
//--- input parameters
input int                InpFastEMA=12;               // Fast EMA period
input int                InpSlowEMA=26;               // Slow EMA period
input int                InpSignalSMA=9;              // Signal SMA period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Applied price
//--- indicator buffers
double                   ExtMacdBuffer[];
double                   ExtSignalBuffer[];
double                   ExtFastMaBuffer[];
double                   ExtSlowMaBuffer[];
//--- MA handles
int                      ExtFastMaHandle;
int                      ExtSlowMaHandle;
//+------------------------------------------------------------------+
```

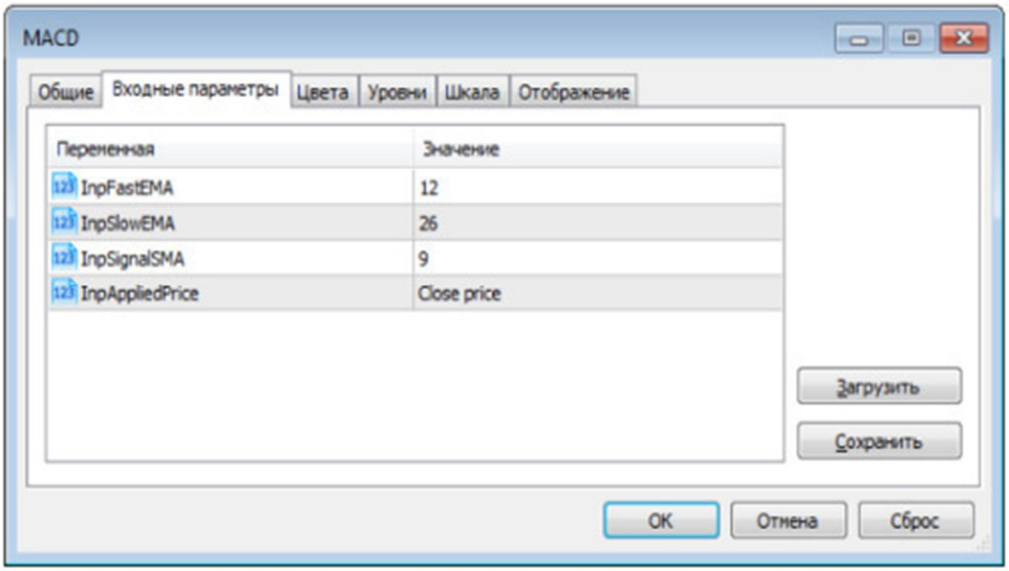
В коде индикатора такие параметры задаются input переменными с модификатором input, который указывается перед типом данных. Как правило, input переменные объявляются сразу после свойств индикатора.

Например, для индикатора MACD – это периоды для

экспоненциальной скользящей средней с коротким периодом от цены, экспоненциальной скользящей средней с длинным периодом от цены, сглаживающей скользящей средней с коротким периодом от разницы двух остальных скользящих, и тип применяемой цены.

Здесь надо отметить то, что в диалоговом окне присоединения индикатора к графику отображаются не имена переменных, а комментарии к ним.

Если убрать комментарии, входные параметры отобразятся следующим образом.



Здесь уже отображаются имена переменных.

Как вы сами, наверное, уже догадались, комментарии используются для отображения, чтобы облегчить пользователю понимание их предназначения.

Здесь также видно, что входными параметрами могут быть не только отдельные переменные, но и перечисления, которые отображаются в виде выпадающих списков.

```
22 //--- input parameters
23 input int                  InpFastEMA=12;              // Fast EMA period
24 input int                  InpSlowEMA=26;              // Slow EMA period
25 input int                  InpSignalSMA=9;             // Signal SMA period
26 input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Applied price
27 //--- indicator buffers
28 double                   ExtMacdBuffer[];
29 double                   ExtSignalBuffer[];
30 double                   ExtFastMaBuffer[];
31 double                   ExtSlowMaBuffer[];
```

Для индикатора MACD используется встроенное перечисление ENUM_APPLIED_PRICE, но можно также определить и свое перечисление.

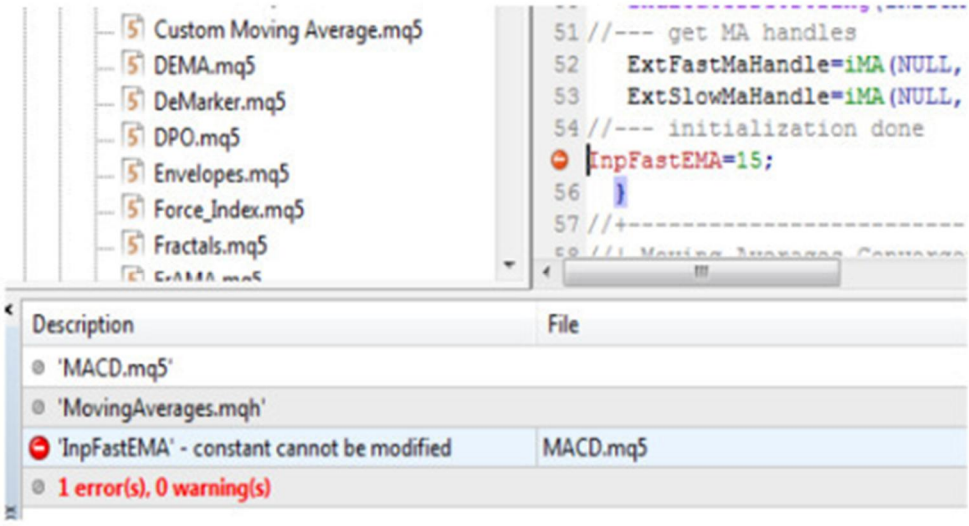
В справочнике приводится соответствующий пример.

```
#property script_show_inputs
//--- day of week
enum dayOfWeek
{
  S=0,    // Sunday
  M=1,    // Monday
  T=2,    // Tuesday
  W=3,    // Wednesday
  Th=4,   // Thursday
  Fr=5,   // Friday,
  St=6,   // Saturday
 };
//--- input parameters
input dayOfWeek swapday=W;
```

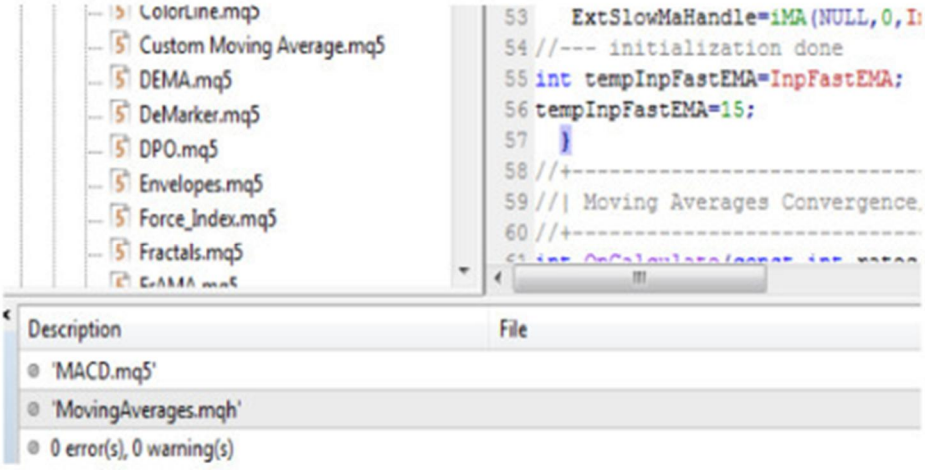
В этом примере команда #property script_show_inputs используется для скриптов, для индикаторов ее можно опустить.

Основное отличие input переменных от других типов переменных состоит в том, что изменить их значение может только пользователь в диалоговом окне индикатора.

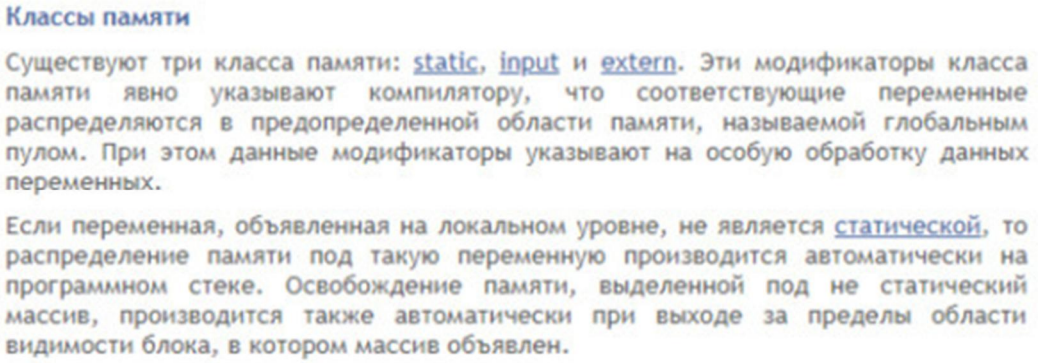
Если в коде индикатора попытаться изменить значение входного параметра, при компиляции возникнет ошибка.



Поэтому, если вы хотите при расчетах использовать измененное значение входного параметра, нужно использовать промежуточную переменную.



Помимо input переменных MQL5-код использует локальные переменные, статические переменные, глобальные переменные и extern переменные.



Классы памяти

Существуют три класса памяти: static, input и extern. Эти модификаторы класса памяти явно указывают компилятору, что соответствующие переменные распределяются в предопределенной области памяти, называемой глобальным пулом. При этом данные модификаторы указывают на особую обработку данных переменных.

Если переменная, объявленная на локальном уровне, не является статической, то распределение памяти под такую переменную производится автоматически на программном стеке. Освобождение памяти, выделенной под не статический массив, производится также автоматически при выходе за пределы области видимости блока, в котором массив объявлен.

С локальными переменными в принципе все понятно, они объявляются в блоке кода, например, в цикле или функции, там же инициализируются, и, после выполнения блока кода, память, выделенная под локальные переменные в программном стеке, освобождается.

Тут особо надо отметить, что для локальных объектов, созданных с помощью оператора new, в конце блока кода нужно применить оператор delete для освобождения памяти.

Глобальные переменные, как правило, объявляются после свойств индикатора, входных параметров и массивов буферов индикатора, перед функциями.

Глобальные переменные видны в пределах всей программы, их значение может быть изменено в любом месте программы и память, выделяемая под глобальные переменные вне программного стека, освобождается при выгрузке программы.

Здесь видно, что input переменные это те же глобальные переменные, за исключением опции – их значение не может быть изменено в любом месте программы.

Если глобальную или локальную переменную объявить со спецификатором const – это так же не позволит изменять значение этой переменной в процессе выполнения программы.

Статические переменные определяются модификатором static, который указывается перед типом данных.

Со статическими переменными все немного сложнее, но легче всего их понять, сравнивая статические переменные с локальными и глобальными переменными.

В принципе, статическая переменная, объявленная там же, где и глобальная переменная, ничем не отличается от глобальной переменной.

Хитрость начинается, если локальную переменную объявить с модификатором static.

В этом случае, после выполнения блока кода, память, выделенная под статическую переменную, не освобождается. И при следующем выполнении того же блока кода, предыдущее значение статической переменной можно использовать.

Хотя область видимости такой статической переменной ограничивается те же самым блоком кода, в котором она была объявлена.

extern переменные это аналог статических глобальных переменных. Нельзя объявить локальную переменную с модификатором extern.

Отличие extern переменных от статических глобальных переменных проще всего продемонстрировать на индикаторе MACD.

```
//+------------------------------------------------------------------+
//|                                                         MACD.mq5 |
//|                  Copyright 2009-2017, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Moving Average Convergence/Divergence"
#include <MovingAverages.mqh>
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_type2   DRAW_LINE
#property indicator_color1  Silver
#property indicator_color2  Red
#property indicator_width1  2
#property indicator_width2  1
#property indicator_label1  "MACD"
#property indicator_label2  "Signal"
```

extern int a=0;

static int a=0;

Индикатор MACD имеет включаемый файл MovingAverages, обозначенный с помощью директивы #include и расположенный в папке Include.

Если в файле MovingAverages и файле MACD одновременно объявить extern-переменную:

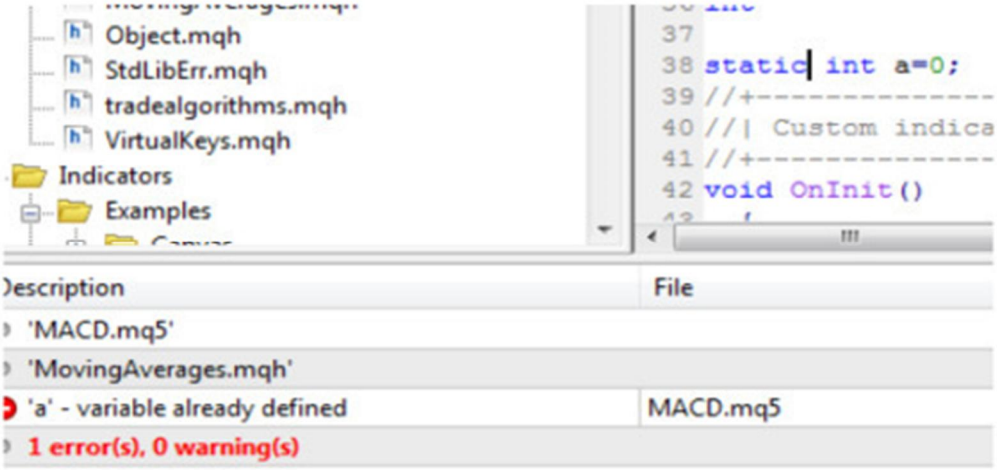
extern int a=0;

то при компиляции обоих файлов все пройдет удачно, и переменную можно будет использовать.

Если же в файле MovingAverages и файле MACD одновременно объявить статическую глобальную переменную:

static int a=0;

тогда при компиляции обоих файлов возникнет ошибка.



Помимо команды #include полезной является также директива #define, которая позволяет делать подстановку выражения вместо идентификатора, например:

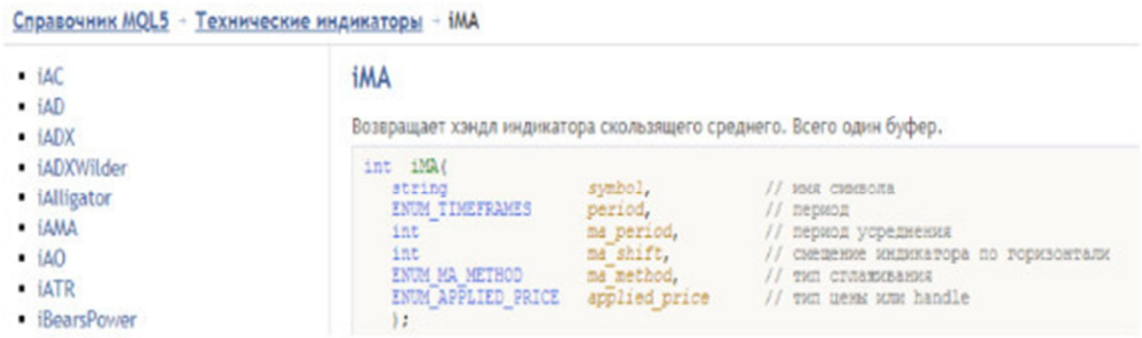
```
#define ABC                 100
#define PI                  3.14
#define COMPANY_NAME        "MetaQuotes Software Corp."
...
void ShowCopyright()
  {
   Print("Copyright  2001-2009, ",COMPANY_NAME);
   Print("https://www.metaquotes.net");
  }
```

#define PI 3.14

Хэндл индикатора

Начнем с цитаты:

HANDLE идентифицирует объект, которым Вы можете манипулировать.
Джеффри РИХТЕР «Windows для профессионалов».



Переменные типа handle представляют собой указатель на некоторую системную структуру или индекс в некоторой системной таблице, которая содержит адрес структуры.

Таким образом, получив хэндл некоторого индикатора, мы можем использовать его данные для построения своего индикатора.

Хэндл индикатора представляет собой переменную типа int и объявляется, как правило, после объявления массивов буферов индикатора, вместе с глобальными переменными, например в индикаторе MACD:

```
//--- indicator buffers
double                  ExtMacdBuffer[];
double                  ExtSignalBuffer[];
double                  ExtFastMaBuffer[];
double                  ExtSlowMaBuffer[];
//--- MA handles
int                     ExtFastMaHandle;
int                     ExtSlowMaHandle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtMacdBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ExtSignalBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,ExtFastMaBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(3,ExtSlowMaBuffer,INDICATOR_CALCULATIONS);
//--- sets first bar from what index will be drawn
   PlotIndexSetInteger(1,PLOT_DRAW_BEGIN,InpSignalSMA-1);
//--- name for Dindicator subwindow label
   IndicatorSetString(INDICATOR_SHORTNAME,"MACD("+string(InpFastEMA)+","+string(InpSlowEMA)+","+string(InpSignalSMA)+")");
//--- get MA handles
   ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
   ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
//--- initialization done
  }
```

Объявляются два хэндла – int ExtFastMaHandle и int ExtSlowMaHandle.

Здесь хэндлы индикаторов – это указатели на индикатор скользящего среднего с разными периодами 12 и 26.

Объявив эти переменные, мы естественно реально ничего не получаем, так как объекта индикатора, данные которого мы хотим использовать, еще не существует.

Создать в глобальном кеше клиентского терминала копию соответствующего технического индикатора и получить ссылку на нее можно несколькими способами.

Если это стандартный индикатор, проще всего получить его хэндл можно с помощью стандартной функции для работы с техническими индикаторами.

iMA

Возвращает хэндл индикатора скользящего среднего. Всего один буфер.

```
int  iMA(
   string               symbol,           // имя символа
   ENUM_TIMEFRAMES      period,           // период
   int                  ma_period,        // период усреднения
   int                  ma_shift,         // смещение индикатора по горизонтали
   ENUM_MA_METHOD       ma_method,        // тип сглаживания
   ENUM_APPLIED_PRICE   applied_price     // тип цены или handle
   );
```

Стандартная функция для индикатора скользящего среднего это функция iMA.

И в индикаторе MACD хэндлы индикатора скользящего среднего получаются с помощью вызова функции iMA в функции OnInit ().

```
49 //--- get MA handles
50    ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
51    ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
```

```
22 //--- input parameters
23 input int                InpFastEMA=12;              // Fast EMA period
24 input int                InpSlowEMA=26;              // Slow EMA period
25 input int                InpSignalSMA=9;             // Signal SMA period
26 input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Applied price
```

где используются свойства индикатора – InpFastEMA, InpSlowEMA и InpAppliedPrice.

Предположим, что мы хотим использовать не стандартный, а пользовательский индикатор.



В папке Indicators/Examples редактора MQL5 есть нужный нам индикатор – это файл Custom Moving Average.mq5.

Для вызова того индикатора воспользуемся функцией iCustom.

iCustom

Возвращает хэндл указанного пользовательского индикатора.

```
int  iCustom(
   string           symbol,     // имя символа
   ENUM_TIMEFRAMES  period,     // период
   string           name        // папка/имя_пользовательского индикатора
   ...                          // список входных параметров индикатора
   );
```

В функции OnInit () индикатора MACD изменим код, где для получения хэндлов вместо стандартной функции, используем функцию iCustom.

```
// ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
// ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
ExtFastMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpFastEMA,0,MODE_EMA,InpAppliedPrice);
ExtSlowMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
```

После компиляции индикатора мы увидим, что его отображение никак не изменилось.



Еще один способ получить хэндл пользовательского индикатора, это использовать функцию IndicatorCreate.

IndicatorCreate

Возвращает хэндл указанного технического индикатора, созданного на основе массива параметров типа MqlParam.

```
int  IndicatorCreate(
   string           symbol,                    // имя символа
   ENUM_TIMEFRAMES  period,                    // период
   ENUM_INDICATOR   indicator_type,            // тип индикатора из перечисления ENUM_INDICATOR
   int              parameters_cnt=0,          // количество параметров
   const MqlParam   parameters_array[]=NULL,   // массив параметров
   );
```

В функции OnInit () индикатора MACD изменим код, где для получения хэндлов используем функцию IndicatorCreate.

```
// ExtFastMaHandle=iMA(NULL,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
// ExtSlowMaHandle=iMA(NULL,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
// ExtFastMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpFastEMA,0,MODE_EMA,InpAppliedPrice);
// ExtSlowMaHandle=iCustom(NULL,0,"Examples\\Custom Moving Average", InpSlowEMA,0,MODE_EMA,InpAppliedPrice);

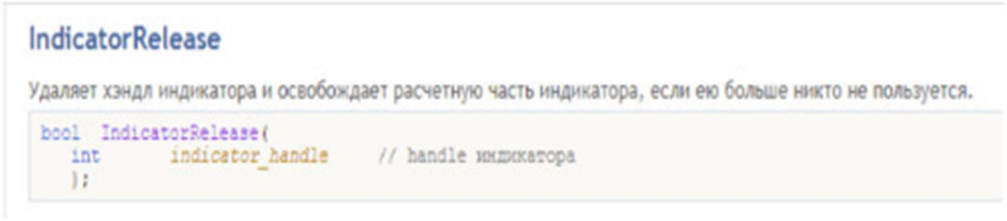
MqlParam params[];
 ArrayResize(params,5);
 params[0].type         =TYPE_STRING;
 params[0].string_value="Examples\\Custom Moving Average";
//--- set ma_period
   params[1].type         =TYPE_INT;
   params[1].integer_value=InpFastEMA;
//--- set ma_shift
   params[2].type         =TYPE_INT;
   params[2].integer_value=0;
//--- set ma_method
   params[3].type         =TYPE_INT;
   params[3].integer_value=MODE_EMA;
//--- set applied_price
   params[4].type         =TYPE_INT;
   params[4].integer_value=InpAppliedPrice;

ExtFastMaHandle=IndicatorCreate(NULL,NULL,IND_CUSTOM,4,params);
params[1].integer_value=InpSlowEMA;
ExtSlowMaHandle=IndicatorCreate(NULL,NULL,IND_CUSTOM,4,params);

```

После компиляции индикатора мы опять увидим, что его отображение никак не изменилось.

После получения хэндла индикатора, если он используется в коде один раз, для экономии памяти неплохо использовать функцию IndicatorRelease.



Которая удаляет хэндл индикатора и освобождает расчетную часть индикатора.

Хорошо, хэндл индикатора мы получили. Как же теперь извлечь его данные?

Делается это в функции OnCalculate с помощью функции CopyBuffer.

Обращение по начальной позиции и количеству требуемых элементов

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   int       start_pos,            // откуда начнем
   int       count,                // сколько копируем
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

Обращение по начальной дате и количеству требуемых элементов

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   datetime  start_time,           // с какой даты
   int       count,                // сколько копируем
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

Обращение по начальной и конечной датам требуемого интервала времени

```
int  CopyBuffer(
   int       indicator_handle,     // handle индикатора
   int       buffer_num,           // номер буфера индикатора
   datetime  start_time,           // с какой даты
   datetime  stop_time,            // по какую дату
   double    buffer[]              // массив, куда будут скопированы данные
   );
```

При этом функция CopyBuffer () распределяет размер принимающего массива под размер копируемых данных.

Напомним, что это работает, если принимающий массив является просто динамическим массивом.

Если же принимающий массив связан с буфером индикатора, тогда клиентский терминал сам заботится о том, чтобы размер такого массива соответствовал количеству баров, доступных индикатору для расчета.

В индикаторе MACD именно такая ситуация.

```
35 //+------------------------------------------------------------------+
36 //| Custom indicator initialization function                         |
37 //+------------------------------------------------------------------+
38 void OnInit()
39   {
40 //--- indicator buffers mapping
41    SetIndexBuffer(0,ExtMacdBuffer,INDICATOR_DATA);
42    SetIndexBuffer(1,ExtSignalBuffer,INDICATOR_DATA);
43    SetIndexBuffer(2,ExtFastMaBuffer,INDICATOR_CALCULATIONS);
44    SetIndexBuffer(3,ExtSlowMaBuffer,INDICATOR_CALCULATIONS);
```

Промежуточные массивы ExtFastMaBuffer и ExtSlowMaBuffer привязаны к буферам индикатора с помощью функции SetIndexBuffer.

```
118 //--- get Fast EMA buffer
119    if(IsStopped()) return(0); //Checking for stop flag
120    if(CopyBuffer(ExtFastMaHandle,0,0,to_copy,ExtFastMaBuffer)<=0)
121      {
122       Print("Getting fast EMA is failed! Error",GetLastError());
123       return(0);
124      }
125 //--- get SlowSMA buffer
126    if(IsStopped()) return(0); //Checking for stop flag
127    if(CopyBuffer(ExtSlowMaHandle,0,0,to_copy,ExtSlowMaBuffer)<=0)
128      {
129       Print("Getting slow SMA is failed! Error",GetLastError());
130       return(0);
131      }
```

И в эти массивы производится копирование буфера индикатора Moving Average на основе его хэндлов с помощью функции CopyBuffer.

```
40 //--- indicator buffers mapping
41    SetIndexBuffer(0,ExtMacdBuffer,INDICATOR_DATA);
42    SetIndexBuffer(1,ExtSignalBuffer,INDICATOR_DATA);
43 //    SetIndexBuffer(2,ExtFastMaBuffer,INDICATOR_CALCULATIONS);
44 //    SetIndexBuffer(3,ExtSlowMaBuffer,INDICATOR_CALCULATIONS);
```

Время	Источник	Сообщение
2018.11.08 11:14:38.678	MACD (EURUSD,H1)	array out of range in 'MACD.mq5' (139,39)

Торговля | Активы | История | Новости 75 | Почта 6 | Календарь | Компания | Маркет | Алерты | Сигналы

```
120    if(CopyBuffer(ExtFastMaHandle,0,0,to_copy,ExtFastMaBuffer)<=0)
121      {
122       Print("Getting fast EMA is failed! Error",GetLastError());
123       return(0);
124      }
```

Если убрать привязку массивов ExtFastMaBuffer и ExtSlowMaBuffer к буферам индикатора, тогда клиентский терминал выдаст ошибку.

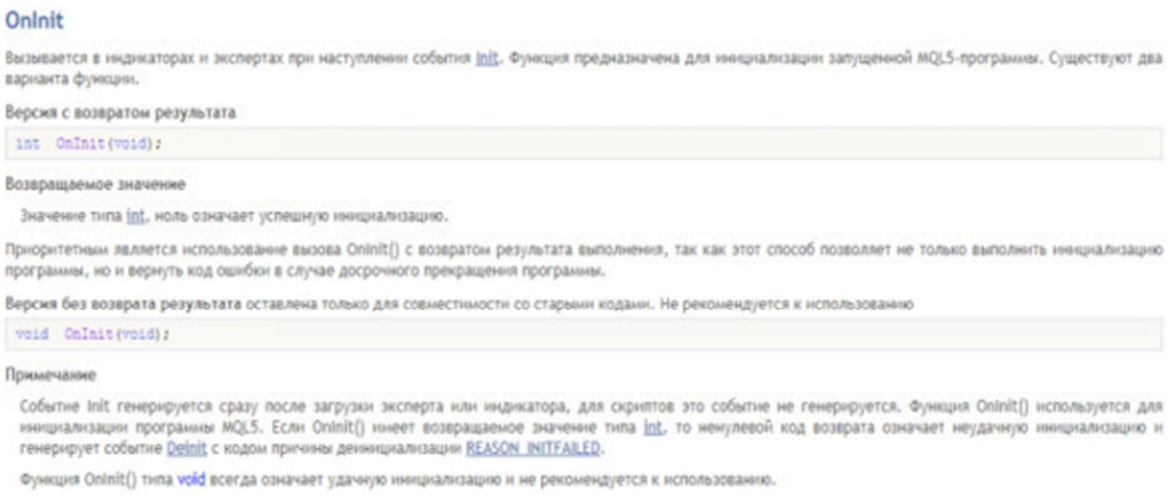
Происходит это потому, что при загрузке индикатора значение to_copy равно размеру ценовой истории, а дальше to_copy=1 и производится частичное копирование в массивы ExtFastMaBuffer и ExtSlowMaBuffer, при этом их размеры становятся равны 1.

В этом случае применением функции ArrayResize проблему не решить, так как функция CopyBuffer все равно будет уменьшать размер массива до 1.

Можно конечно использовать еще один массив-посредник, в который копировать один элемент. И уже из этого массива-посредника производить копирование в промежуточный массив, но проще всего, конечно, просто привязать промежуточный массив к буферу индикатора.

Функция OnInit

Как уже говорилось, функции OnInit (), OnDeinit (), OnCalculate () вызываются клиентским терминалом при наступлении определенных событий.



OnInit

Вызывается в индикаторах и экспертах при наступлении события Init. Функция предназначена для инициализации запущенной MQL5-программы. Существуют два варианта функции.

Версия с возвратом результата

```
int  OnInit(void);
```

Возвращаемое значение

Значение типа int, ноль означает успешную инициализацию.

Приоритетным является использование вызова OnInit() с возвратом результата выполнения, так как этот способ позволяет не только выполнить инициализацию программы, но и вернуть код ошибки в случае досрочного прекращения программы.

Версия без возврата результата оставлена только для совместимости со старыми кодами. Не рекомендуется к использованию

```
void  OnInit(void);
```

Примечание

Событие Init генерируется сразу после загрузки эксперта или индикатора, для скриптов это событие не генерируется. Функция OnInit() используется для инициализации программы MQL5. Если OnInit() имеет возвращаемое значение типа int, то ненулевой код возврата означает неудачную инициализацию и генерирует событие Deinit с кодом причины деинициализации REASON_INITFAILED.

Функция OnInit() типа void всегда означает удачную инициализацию и не рекомендуется к использованию.

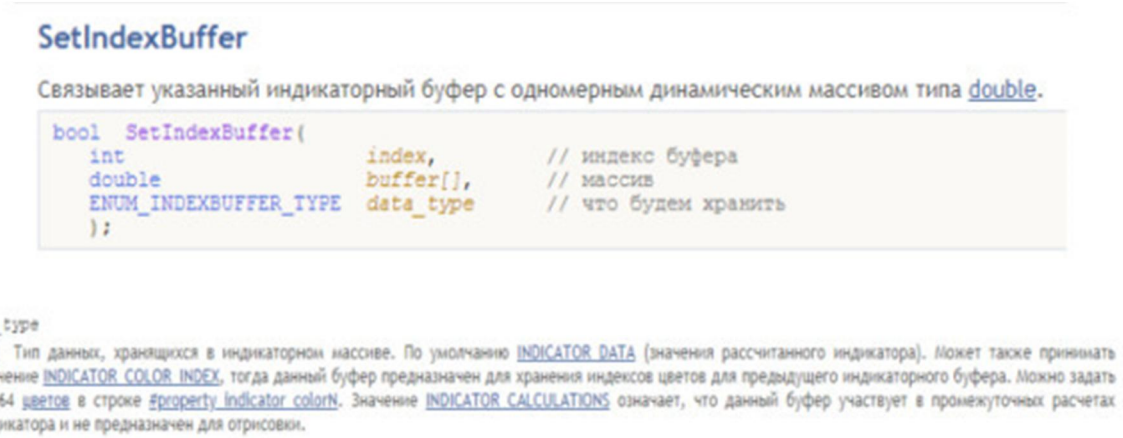
Функция OnInit () вызывается сразу после загрузки индикатора и соответственно используется для его инициализации.

Инициализация индикатора включает в себя привязку массивов к буферам индикатора, инициализацию глобальных переменных,

включая инициализацию хэндлеров используемых индикаторов, а также программную установку свойств индикатора.

Давайте разберем некоторые из этих пунктов более подробно.

Как уже было показано, привязка массивов к буферам индикатора осуществляется с помощью функции SetIndexBuffer.



SetIndexBuffer

Связывает указанный индикаторный буфер с одномерным динамическим массивом типа double.

```
bool  SetIndexBuffer(
   int                  index,        // индекс буфера
   double               buffer[],     // массив
   ENUM_INDEXBUFFER_TYPE data_type    // что будем хранить
   );
```

data_type

[in] Тип данных, хранящихся в индикаторном массиве. По умолчанию INDICATOR_DATA (значения рассчитанного индикатора). Может также принимать значение INDICATOR_COLOR_INDEX, тогда данный буфер предназначен для хранения индексов цветов для предыдущего индикаторного буфера. Можно задать до 64 цветов в строке #property indicator_colorN. Значение INDICATOR_CALCULATIONS означает, что данный буфер участвует в промежуточных расчетах индикатора и не предназначен для отрисовки.

Где data_type может быть INDICATOR_DATA (данные индикатора для отрисовки, по умолчанию, можно не указывать), INDICATOR_COLOR_INDEX (цвет индикатора), INDICATOR_CALCULATIONS (буфер промежуточных расчетов индикатора).

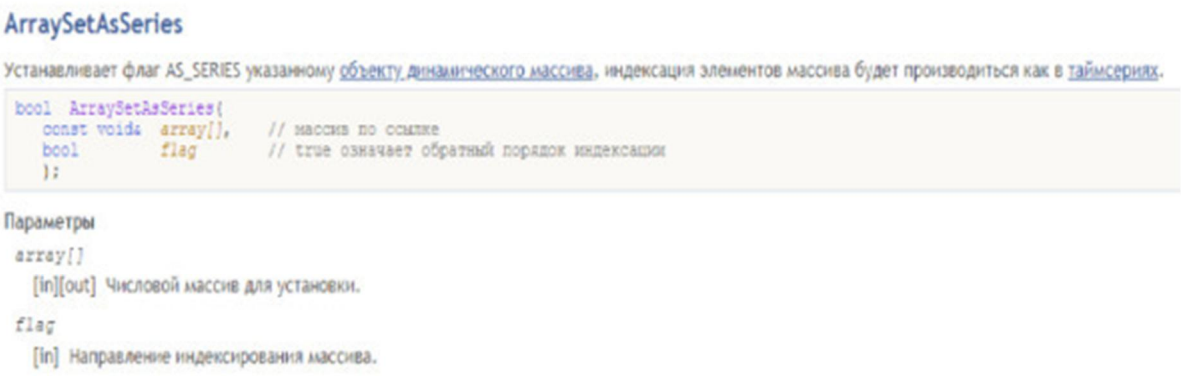
После применения функции SetIndexBuffer к динамическому массиву, его размер автоматически поддерживается равным количеству баров, доступных индикатору для расчета.

Каждый индекс массива типа INDICATOR_COLOR_INDEX соответствует индексу массива типа INDICATOR_DATA, а значение индекса массива типа INDICATOR_COLOR_INDEX определяет цвет отображения индекса массива типа INDICATOR_DATA.

Значение индекса массива типа INDICATOR_COLOR_INDEX, при его установке, берется из свойства #property indicator_colorN как индекс цвета в строке.

Индекс буфера типа INDICATOR_COLOR_INDEX должен следовать за индексом буфера типа INDICATOR_DATA.

После привязки динамического массива к буферу индикатора можно поменять порядок доступа к массиву от конца к началу, т.е. значение массива с индексом 0 будет соответствовать последнему полученному значению индикатора. Сделать это можно с помощью функции ArraySetAsSeries.



ArraySetAsSeries

Устанавливает флаг AS_SERIES указанному объекту динамического массива, индексация элементов массива будет производиться как в таймсериях.

```
bool  ArraySetAsSeries(
   const void&  array[],    // массив по ссылке
   bool         flag        // true означает обратный порядок индексации
   );
```

Параметры

array[]

[in][out] Числовой массив для установки.

flag

[in] Направление индексирования массива.

При применении функции ArraySetAsSeries физическое хранение данных массива не меняется, в памяти, массив, как и прежде, хранится в порядке от первого значения до последнего значения.

Функция ArraySetAsSeries меняет лишь программный доступ к элементам массива – от последнего элемента массива к первому элементу массива.

В функции OnInit () также может осуществляться проверка входных параметров на корректность, так как пользователь может ввести все, что угодно.

При этом значение входного параметра переназначается с помощью глобальной переменной, и далее в расчетах используется уже значение глобальной переменной.

Например, для индикатора ADX это выглядит так:

```mql5
void OnInit()
  {
//--- check for input parameters
   if(InpPeriodADX>=100 || InpPeriodADX<=0)
     {
      ExtADXPeriod=14;
      printf("Incorrect value for input variable Period_ADX=%d. Indicator will use value=%d for calculations.",
      InpPeriodADX,ExtADXPeriod);
     }
   else ExtADXPeriod=InpPeriodADX;
```

здесь ExtADXPeriod – глобальная переменная, а InpPeriodADX – входной параметр.

При использовании хэндлов индикатора, можно указывать символ (финансовый инструмент) для которого индикатор будет создаваться.

При этом такой символ может определяться пользователем.

В функции OnInit () также полезно проверить этот входной параметр на корректность.

```mql5
//-----------------------------------
input string symbol=" ";  // символ
string name=symbol;
```

```mql5
//--- удалим пробелы слева и справа
   StringTrimRight(name);
   StringTrimLeft(name);
//--- если после этого длина строки name нулевая
   if(StringLen(name)==0)
     {
      //--- возьмем символ с графика, на котором запущен индикатор
      name=_Symbol;
     }

ExtFastMaHandle=iMA(name,0,InpFastEMA,0,MODE_EMA,InpAppliedPrice);
ExtSlowMaHandle=iMA(name,0,InpSlowEMA,0,MODE_EMA,InpAppliedPrice);
```

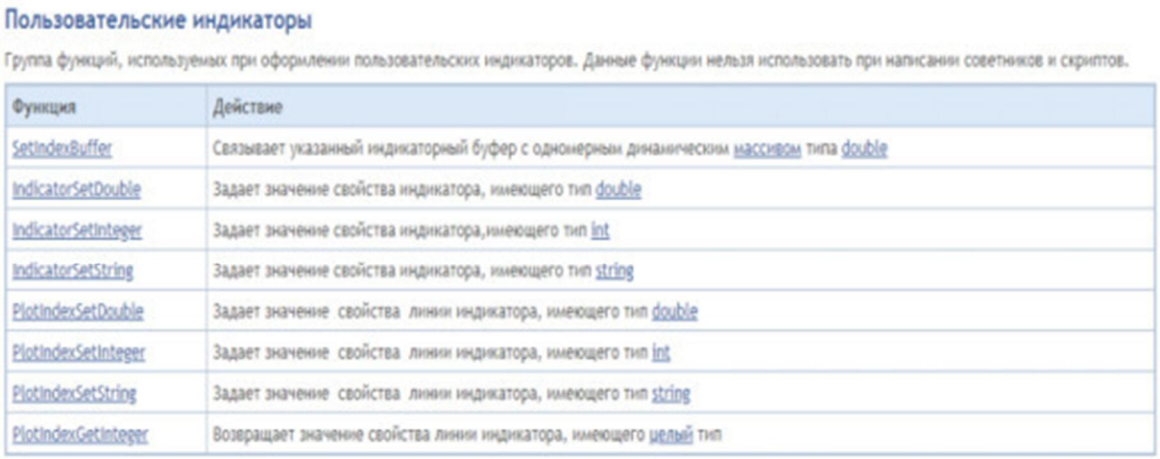
Например, в коде индикатора MACD пусть определен входной параметр:

input string symbol=«»;

Объявим глобальную переменную:

string name=symbol;

И в функции OnInit () произведем проверку – удалим пробелы слева и справа с помощью функции StringTrimRight, и если после этого длина строки name нулевая, возьмем символ с графика, на котором запущен индикатор.



Пользовательские индикаторы

Группа функций, используемых при оформлении пользовательских индикаторов. Данные функции нельзя использовать при написании советников и скриптов.

Функция	Действие
SetIndexBuffer	Связывает указанный индикаторный буфер с одномерным динамическим массивом типа double
IndicatorSetDouble	Задает значение свойства индикатора, имеющего тип double
IndicatorSetInteger	Задает значение свойства индикатора,имеющего тип int
IndicatorSetString	Задает значение свойства индикатора, имеющего тип string
PlotIndexSetDouble	Задает значение свойства линии индикатора, имеющего тип double
PlotIndexSetInteger	Задает значение свойства линии индикатора, имеющего тип int
PlotIndexSetString	Задает значение свойства линии индикатора, имеющего тип string
PlotIndexGetInteger	Возвращает значение свойства линии индикатора, имеющего целый тип

Программная установка свойств индикатора осуществляется с помощью функций IndicatorSetDouble, IndicatorSetInteger, IndicatorSetString, PlotIndexSetDouble, PlotIndexSetInteger, PlotIndexSetString.

IndicatorSetDouble

Задаёт значение соответствующего свойства индикатора. Свойство индикатора должно быть типа double.

Вызов с указанием идентификатора свойства.

```
bool  IndicatorSetDouble(
   int     prop_id,          // идентификатор
   double  prop_value        // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  IndicatorSetDouble(
   int     prop_id,          // идентификатор
   int     prop_modifier,    // модификатор
   double  prop_value        // устанавливаемое значение
   )
```

IndicatorSetDouble(INDICATOR_LEVELVALUE, 0, 50)

property indicator_level1 50

Функция IndicatorSetDouble позволяет программным способом определять такие свойства индикатора как indicator_minimum, indicator_maximum и indicator_levelN, например:

IndicatorSetDouble (INDICATOR_LEVELVALUE, 0, 50)

является аналогом:

property indicator_level1 50

IndicatorSetInteger

Задает значение соответствующего свойства индикатора. Свойство индикатора должно быть типа int или color.

Вызов с указанием идентификатора свойства.

```
bool  IndicatorSetInteger(
   int  prop_id,          // идентификатор
   int  prop_value        // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  IndicatorSetInteger(
   int  prop_id,          // идентификатор
   int  prop_modifier,    // модификатор
   int  prop_value        // устанавливаемое значение
   )
```

Функция IndicatorSetInteger позволяет программным способом определять такие свойства индикатора как indicator_height, indicator_levelcolor, indicator_levelwidth, indicator_levelstyle.

При этом для уровней необходимо определить их количество, используя функцию IndicatorSetInteger. Например, для индикатора RSI это выглядит следующим образом.

```
//--- indicator settings
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//#property indicator_level1 30
//#property indicator_level2 70
#property indicator_buffers 3
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  DodgerBlue

//#property indicator_levelcolor Red
//#property indicator_levelstyle STYLE_SOLID
//#property indicator_levelwidth 1
```

```
IndicatorSetInteger(INDICATOR_DIGITS,2);
```

```
IndicatorSetInteger(INDICATOR_LEVELS,2);
IndicatorSetDouble(INDICATOR_LEVELVALUE,0,30);
IndicatorSetDouble(INDICATOR_LEVELVALUE,1,70);
IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,0xff0);
IndicatorSetInteger(INDICATOR_LEVELCOLOR,1,0xff0);
IndicatorSetInteger(INDICATOR_LEVELSTYLE,0,STYLE_SOLID);
IndicatorSetInteger(INDICATOR_LEVELSTYLE,1,STYLE_SOLID);
IndicatorSetInteger(INDICATOR_LEVELWIDTH,0,1);
IndicatorSetInteger(INDICATOR_LEVELWIDTH,1,1);
```

Свойства индикатора, связанные с уровнями, заменяем на код, используя функцию IndicatorSetInteger.

Функция IndicatorSetInteger также позволяет определить точность индикатора, например:

IndicatorSetInteger (INDICATOR_DIGITS,2);

В результате будут отображаться только два знака после запятой значения индикатора.

Для функции IndicatorSetString нет соответствующих ей свойств индикатора property.

IndicatorSetString

Задает значение соответствующего свойства индикатора. Свойство индикатора должно быть типа string.

Вызов с указанием идентификатора свойства.

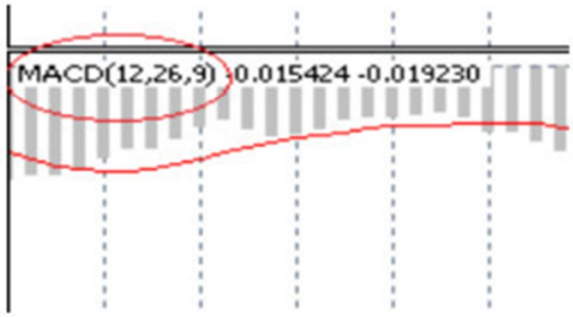
```
bool  IndicatorSetString(
   int     prop_id,          // идентификатор
   string  prop_value        // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  IndicatorSetString(
   int     prop_id,          // идентификатор
   int     prop_modifier,    // модификатор
   string  prop_value        // устанавливаемое значение
   )
```

С помощью функции IndicatorSetString можно определить короткое наименование индикатора, например для индикатора MACD:

```
50 //--- name for Dindicator subwindow label
51   IndicatorSetString(INDICATOR_SHORTNAME,"MACD("+string(InpFastEMA)+","+string(InpSlowEMA)+","+string(InpSignalSMA)+")");
```



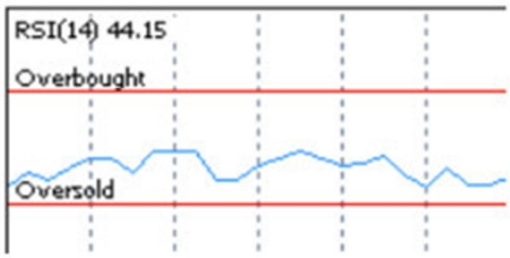
Это выглядит следующим образом.

И соответственно имя индикатора будет отображаться в окне индикатора как:

MACD (12, 26, 9)

Кроме того, функция IndicatorSetString позволяет установить подписи к уровням индикатора, например для индикатора RSI:

```
IndicatorSetString(INDICATOR_LEVELTEXT,
0,"Oversold");
IndicatorSetString(INDICATOR_LEVELTEXT,
1,"Overbought");
```



Можно отобразить подписи к уровням Oversold и Overbought.

PlotIndexSetDouble

Задает значение соответствующего свойства соответствующей линии индикатора. Свойство индикатора должно быть типа double.

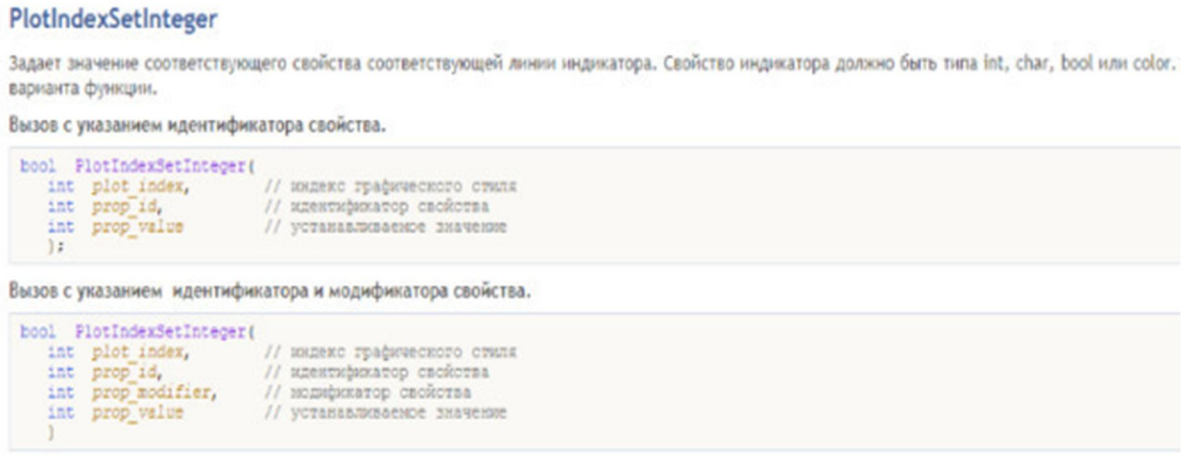
```
bool  PlotIndexSetDouble(
   int     plot_index,     // индекс графического стиля
   int     prop_id,        // идентификатор свойства
   double  prop_value      // устанавливаемое значение
   );
```

PlotIndexSetDouble(индекс_построения,PLOT_EMPTY_VALUE,0);

С помощью функции PlotIndexSetDouble определяют, какое значение буфера индикатора является пустым и не участвует в отрисовке диаграммы индикатора.

Диаграмма индикатора рисуется от одного непустого значения до другого непустого значения индикаторного буфера, пустые значения пропускаются. Чтобы указать, какое значение следует считать «пустым», необходимо определить это значение в свойстве PLOT_EMPTY_VALUE. Например, если индикатор должен рисоваться по ненулевым значениям, то нужно задать нулевое значение в качестве пустого значения буфера индикатора:

PlotIndexSetDouble (индекс_построения, PLOT_EMPTY_VALUE,0);


PlotIndexSetInteger

Задает значение соответствующего свойства соответствующей линии индикатора. Свойство индикатора должно быть типа int, char, bool или color. варианта функции.

Вызов с указанием идентификатора свойства.

```
bool  PlotIndexSetInteger(
   int  plot_index,        // индекс графического стиля
   int  prop_id,           // идентификатор свойства
   int  prop_value         // устанавливаемое значение
   );
```

Вызов с указанием идентификатора и модификатора свойства.

```
bool  PlotIndexSetInteger(
   int  plot_index,        // индекс графического стиля
   int  prop_id,           // идентификатор свойства
   int  prop_modifier,     // модификатор свойства
   int  prop_value         // устанавливаемое значение
   )
```

Функция PlotIndexSetInteger позволяет программным способом, динамически, задавать такие свойства диаграммы индикатора, как код стрелки для стиля DRAW_ARROW, смещение стрелок по вертикали для стиля DRAW_ARROW, количество начальных баров без отрисовки и значений в Окне Данных, тип графического построения, признак отображения значений построения в Окне Данных, сдвиг графического построения индикатора по оси времени в барах, стиль линии отрисовки, толщина линии отрисовки, количество цветов, индекс буфера, содержащего цвет отрисовки.

Давайте разберем каждое из этих свойств по порядку на примере индикатора Custom Moving Average.

```
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
#property indicator_type1   DRAW_ARROW
#property indicator_color1  Red
```

```
PlotIndexSetInteger(0,PLOT_ARROW,2);
```



Изменим свойство indicator_type1 индикатора Custom Moving Average:

#property indicator_type1 DRAW_ARROW

В функции OnInit () добавим вызов функции PlotIndexSetInteger, определяя различный код стрелки для стиля DRAW_ARROW:

PlotIndexSetInteger (0,PLOT_ARROW,2);

В результате получим соответствующий вид стрелки.

PlotIndexSetInteger(0,PLOT_ARROW,3);



PlotIndexSetInteger(0,PLOT_ARROW,4);



PlotIndexSetInteger(0,PLOT_ARROW,5);



PlotIndexSetInteger(0,PLOT_ARROW,6);



Со значением 3 свойства PLOT_ARROW получим другую стрелку.

И так далее, меняя значение свойства, мы будет видеть разные стрелки.

```
154   PlotIndexSetInteger(0,PLOT_ARROW,7);
155   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,100);
```



В функции OnInit () добавим вызов функции PlotIndexSetInteger, определяя смещение стрелок по вертикали для стиля DRAW_ARROW:

PlotIndexSetInteger (0,PLOT_ARROW_SHIFT,100);

В результате диаграмма индикатора сдвинулась вниз.

```
131 void OnInit()
132   {
133 //--- indicator buffers mapping
134    SetIndexBuffer(0,ExtLineBuffer,INDICATOR_DATA);
135 //--- set accuracy
136    IndicatorSetInteger(INDICATOR_DIGITS,_Digits+1);
137 //--- sets first bar from what index will be drawn
138    PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,InpMAPeriod);
139 //---- line shifts when drawing
140    PlotIndexSetInteger(0,PLOT_SHIFT,InpMAShift);
```

В индикаторе Custom Moving Average для определения количества начальных баров без отрисовки используется вызов функции PlotIndexSetInteger:

PlotIndexSetInteger (0,PLOT_DRAW_BEGIN, InpMAPeriod);

где InpMAPeriod – период скользящей средней.

```
 9 //--- indicator settings
10 #property indicator_chart_window
11 #property indicator_buffers 1
12 #property indicator_plots   1
13 //#property indicator_type1   DRAW_ARROW
14 #property indicator_color1  Red
```

```
131 void OnInit()
132   {
133   PlotIndexSetInteger(0, PLOT_DRAW_TYPE, DRAW_ARROW);
```

Идентификатор свойства PLOT_DRAW_TYPE функции PlotIndexSetInteger позволяет программным способом задать свойство индикатора indicator_typeN, например:

PlotIndexSetInteger (0, PLOT_DRAW_TYPE, DRAW_ARROW);

Причем, если одновременно задано свойство indicator_typeN и сделан вызов функции PlotIndexSetInteger с идентификатором PLOT_DRAW_TYPE – действовать будет тип диаграммы, заданный функцией PlotIndexSetInteger.

```
141 //---- line shifts when drawing
142    PlotIndexSetInteger(0,PLOT_SHIFT,InpMAShift);
143
144    PlotIndexSetInteger(0, PLOT_SHOW_DATA, false);
```

Окно данных	
Date	2018.11.08
Time	03:00
Open	1.14324
High	1.14344
Low	1.14299
Close	1.14323
Volume	0
Tick Volume	1760
Spread	11

InpMAShift=0



InpMAShift=10



Убрать отображение текущих значений диаграммы индикатора при наведении курсора мышки в Окне Данных можно с помощью вызова функции PlotIndexSetInteger с идентификатором PLOT_SHOW_DATA.

PlotIndexSetInteger (0, PLOT_SHOW_DATA, false);

В индикаторе Custom Moving Average для определения сдвига графического построения индикатора по оси времени в барах используется вызов функции PlotIndexSetInteger:

PlotIndexSetInteger (0,PLOT_SHIFT, InpMAShift);

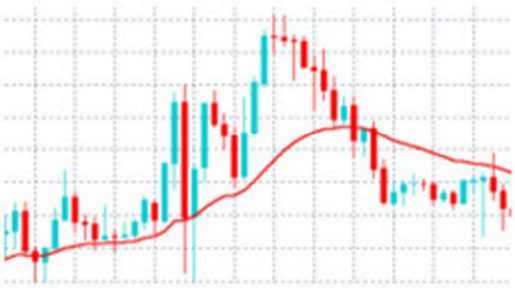
Например, при InpMAShift=10, здесь виден сдвиг индикатора по оси времени.

Такой сдвиг делается для имитации предсказательности индикатора.

```
PlotIndexSetInteger(0, PLOT_DRAW_TYPE, DRAW_LINE);
PlotIndexSetInteger(0, PLOT_LINE_STYLE, STYLE_DASHDOT);
```



```
PlotIndexSetInteger(0, PLOT_LINE_WIDTH, 2);
```



Идентификатор свойства PLOT_LINE_STYLE функции PlotIndexSetInteger позволяет программным способом задать свойство индикатора indicator_styleN, стиль линии отрисовки, например:

PlotIndexSetInteger (0, PLOT_LINE_STYLE, STYLE_DASHDOT);

Идентификатор свойства PLOT_LINE_WIDTH функции PlotIndexSetInteger позволяет программным способом задать свойство индикатора indicator_widthN, толщину линии отрисовки, например:

PlotIndexSetInteger (0, PLOT_LINE_WIDTH, 2);

```
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_type2   DRAW_LINE
//#property indicator_color1  Silver
//#property indicator_color2  Red
#property indicator_width1  2
#property indicator_width2  1
#property indicator_label1  "MACD"
#property indicator_label2  "Signal"
```

```
void OnInit()
  {
PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,1);
PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,Silver);
PlotIndexSetInteger(1,PLOT_COLOR_INDEXES,1);
PlotIndexSetInteger(1,PLOT_LINE_COLOR,0,Red);
```

Программным способом задать свойство индикатора indicator_colorN позволяет вызов функции PlotIndexSetInteger с идентификаторами PLOT_COLOR_INDEXES и PLOT_LINE_COLOR, например, в случае индикатора MACD:

#property indicator_color1 Silver

Равнозначно

PlotIndexSetInteger (0,PLOT_COLOR_INDEXES,1);

PlotIndexSetInteger (0,PLOT_LINE_COLOR,0,Silver);

```
#property indicator_label1  "MACD"
#property indicator_label2  "Signal"
```

```
PlotIndexSetString(0, PLOT_LABEL,
"MACD");
PlotIndexSetString(1, PLOT_LABEL,
"Signal");
```

Функция PlotIndexSetString позволяет программным способом задать свойство индикатора indicator_labelN. Например, для индикатора MACD это будет выглядеть следующим образом:

#property indicator_label1 «MACD»

#property indicator_label2 «Signal»

Это равнозначно

PlotIndexSetString (0, PLOT_LABEL, «MACD»);

PlotIndexSetString (1, PLOT_LABEL, «Signal»);

Рассмотренные выше функции программной установки свойств индикатора можно конечно вызывать и в функции обратного вызова OnCalculate, но глубокого смысла в этом нет, так как они не могут быть применены только к части диаграммы индикатора – они применяются

сразу ко всей диаграмме индикатора. Поэтому для экономии ресурсов лучше всего вызывать эти функции в функции обратного вызова OnInit ().

Функция OnDeinit

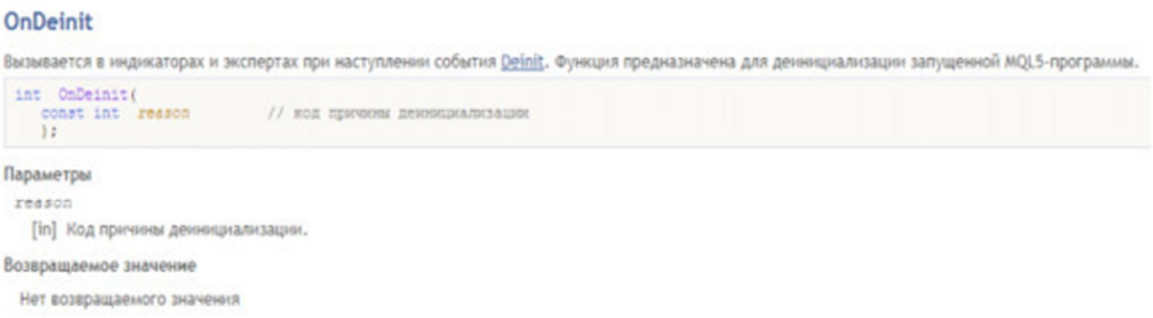
Процитируем справочник:

Событие Deinit генерируется для экспертов и индикаторов в следующих случаях:

– перед переинициализацией в связи со сменой символа или периода графика, к которому прикреплена mql5-программа;

– перед переинициализацией в связи со сменой входных параметров;

– перед выгрузкой mql5-программы.



OnDeinit

Вызывается в индикаторах и экспертах при наступлении события Deinit. Функция предназначена для деинициализации запущенной MQL5-программы.

```
int  OnDeinit(
   const int  reason        // код причины деинициализации
   );
```

Параметры

reason

[in] Код причины деинициализации.

Возвращаемое значение

Нет возвращаемого значения

Так как функция OnDeinit () вызывается при деинициализации, то ее основное предназначение, это освобождение занимаемых ресурсов.

Под освобождением занимаемых ресурсов для индикатора подразумевается очищение графика символа от дополнительных графических объектов.

То есть помимо диаграммы индикатора, мы можем присоединять к графику символа различные объекты – линии, графические фигуры треугольник, прямоугольник и эллипс, знаки, подписи и др. Об этом мы поговорим позже.

Соответственно при деинициализации индикатора было бы неплохо все это убрать с графика символа.

```
void Comment(
  argument,     // первое значение                    Comment("");
  ...           // последующие значения
  );
```

```
bool ObjectDelete(
  long   chart_id,    // chart identifier
  string name         // object name                  ObjectDelete(0,label_name);
  );
```

Первым делом здесь используется функция Comment, которая выводит комментарий, определенный пользователем, в левый верхний угол графика.

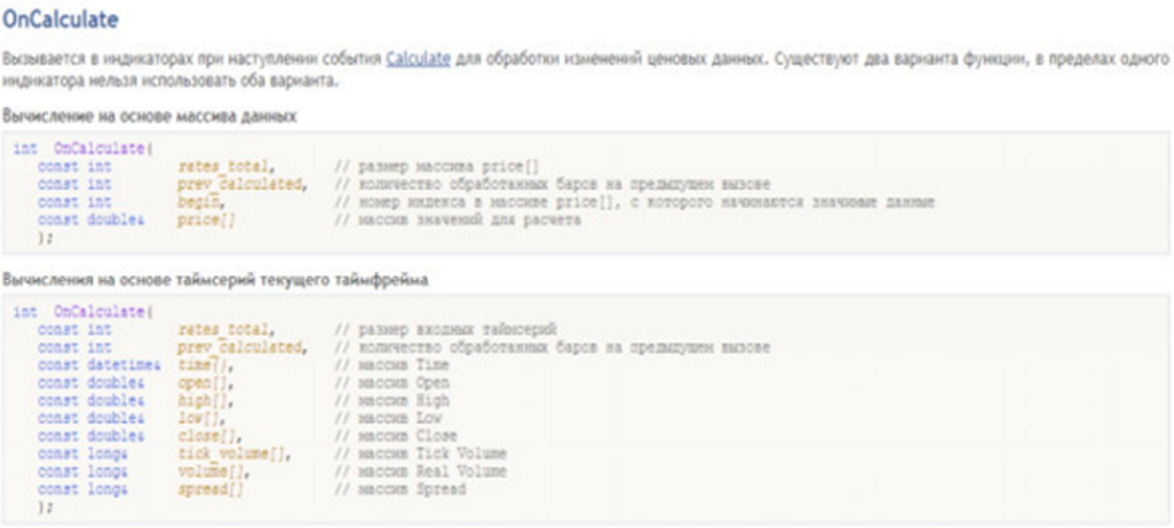
Для очистки от комментариев используются пустые комментарии.

Далее используется функция ObjectDelete, которая удаляет объект с указанным именем с указанного графика.

Позже мы продемонстрируем применение этих функций.

Функция OnCalculate

Функция OnCalculate () вызывается клиентским терминалом при поступлении нового тика по символу, для которого рассчитывается индикатор.



OnCalculate

Вызывается в индикаторах при наступлении события Calculate для обработки изменений ценовых данных. Существуют два варианта функции, в пределах одного индикатора нельзя использовать оба варианта.

Вычисление на основе массива данных

```
int  OnCalculate(
   const int        rates_total,      // размер массива price[]
   const int        prev_calculated,  // количество обработанных баров на предыдущем вызове
   const int        begin,            // номер индекса в массиве price[], с которого начинаются значимые данные
   const double&    price[]           // массив значений для расчета
   );
```

Вычисления на основе таймсерий текущего таймфрейма

```
int  OnCalculate(
   const int        rates_total,      // размер входных таймсерий
   const int        prev_calculated,  // количество обработанных баров на предыдущем вызове
   const datetime&  time[],           // массив Time
   const double&    open[],           // массив Open
   const double&    high[],           // массив High
   const double&    low[],            // массив Low
   const double&    close[],          // массив Close
   const long&      tick_volume[],    // массив Tick Volume
   const long&      volume[],         // массив Real Volume
   const long&      spread[]          // массив Spread
   );
```

Хотя функция OnCalculate () имеет два вида – для индикатора, который может быть рассчитан на основе только одной из ценовых таймсерий, и для индикатора, который рассчитывается с использованием нескольких ценовых таймсерий.

Здесь мы будем пользоваться полной версией функции OnCalculate () как наиболее гибкой и предоставляющей наибольшие возможности.

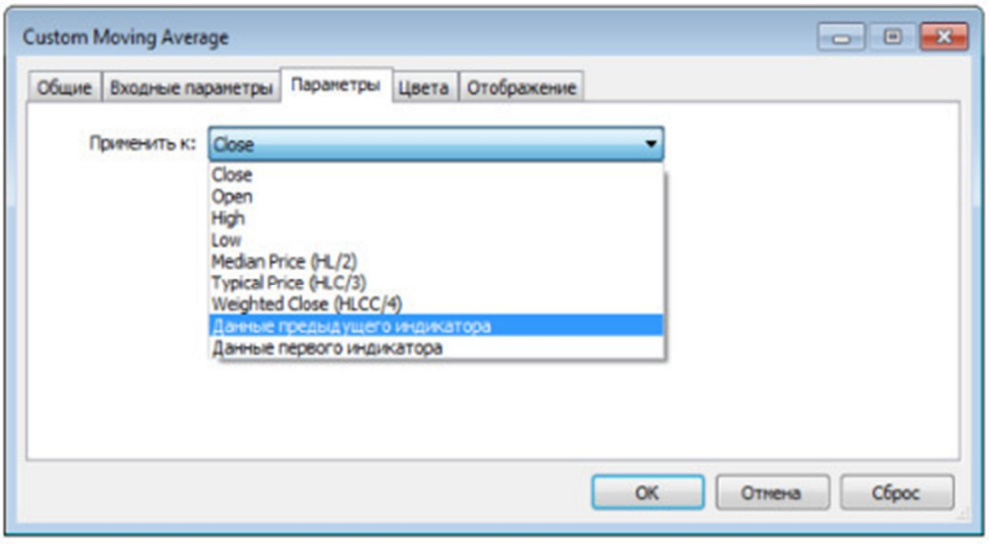
Единственное, что мы должны отметить об усеченной функции OnCalculate (), это то, что она имеет опцию использования в качестве массива price [] рассчитанного буфера другого индикатора.

Продемонстрируем это на примере индикатора MACD и индикатора Custom Moving Average, который использует как раз усеченную функцию OnCalculate ().



Присоединим сначала индикатор MACD к графику символа, а затем перетащим индикатор MA из папки Индикаторы – Трендовые в окно индикатора MACD.

Затем еще перетащим индикатор Custom Moving Average в окно индикатора MACD, при этом откроется окно параметров индикатора Custom Moving Average:

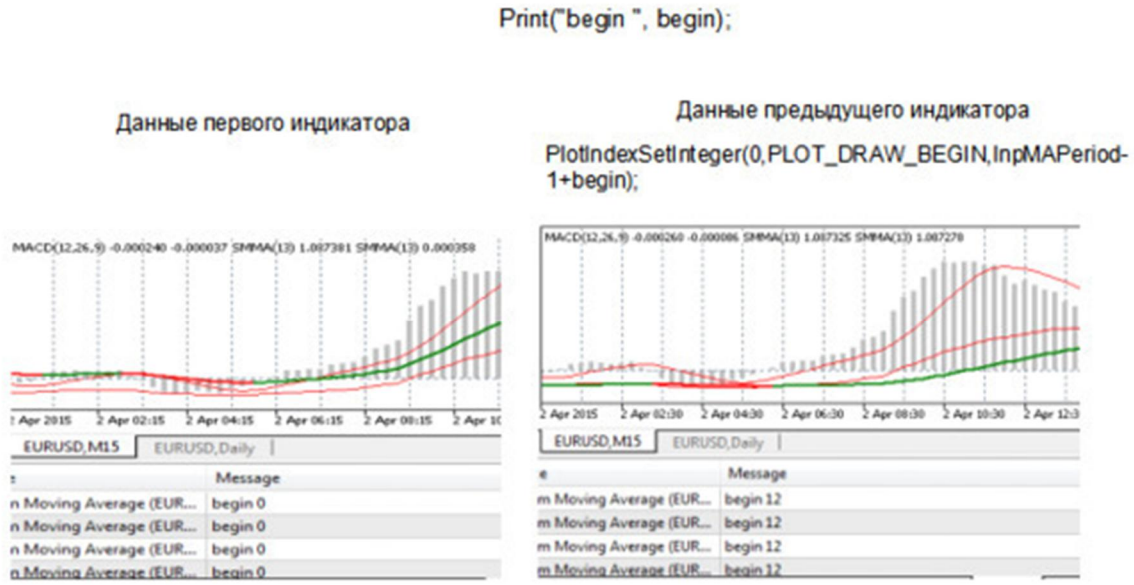


В списке выбора – что использовать в качестве массива price [] – будут пункты:

Данные предыдущего индикатора

Данные первого индикатора

Здесь пункт Данные первого индикатора означает, что в качестве массива price [] будет использоваться массив ExtMacdBuffer буфера индикатора MACD, а пункт Данные предыдущего индикатора означает, что в качестве массива price [] будет использоваться массив ExtLineBuffer буфера индикатора MA.



Если в функцию OnCalculate индикатора Custom Moving Average добавить:

Print («begin», begin);

То при выборе Данные первого индикатора будет выводиться:

Begin 0

А при выборе Данные предыдущего индикатора будет выводиться:

Begin 12

В первом случае, begin=0, так как для буфера ExtMacdBuffer индикатора MACD функция PlotIndexSetInteger с параметром PLOT_DRAW_BEGIN не вызывается.

А во втором случае, begin=12, так как для буфера ExtLineBuffer индикатора MA вызывается функция PlotIndexSetInteger:

В ней говорится о том, что массив буфера ExtLineBuffer индикатора MA заполняется, начиная с InpMAPeriod-1 бара, соответственно значение переменной begin функции OnCalculate индикатора Custom Moving

Average будет также равно InpMAPeriod-1.

Вычисления на основе таймсерий текущего таймфрейма

```
int  OnCalculate(
   const int        rates_total,      // размер входных таймсерий
   const int        prev_calculated,  // количество обработанных баров на предыдущем вызове
   const datetime&  time[],           // массив Time
   const double&    open[],           // массив Open
   const double&    high[],           // массив High
   const double&    low[],            // массив Low
   const double&    close[],          // массив Close
   const long&      tick_volume[],    // массив Tick Volume
   const long&      volume[],         // массив Real Volume
   const long&      spread[]          // массив Spread
   );
```

Вернемся к полной версии функции OnCalculate ().

Как правило, код функции OnCalculate () проектируется таким образом, чтобы при загрузке индикатора и первом вызове функции OnCalculate (), буфера индикатора были рассчитаны на основе всей загруженной ценовой истории, а при последующем поступлении нового тика и вызове функции OnCalculate (), рассчитывалось бы только одно новое значение, которое добавляется в конец массива буфера индикатора.

Но в начале кода функции OnCalculate () нужно конечно проверить, достаточный ли размер ценовой истории был загружен при загрузке индикатора.

Для этого проверяется значение переменной rates_total – размер входных таймсерий.

ADX

```
80 //+------------------------------------------------------------------+
81 //| Custom indicator iteration function                              |
82 //+------------------------------------------------------------------+
83 int OnCalculate(const int rates_total,
84                  const int prev_calculated,
85                  const datetime &time[],
86                  const double &open[],
87                  const double &high[],
88                  const double &low[],
89                  const double &close[],
90                  const long &tick_volume[],
91                  const long &volume[],
92                  const int &spread[])
93   {
94 //--- checking for bars count
95    if(rates_total<ExtADXPeriod)
96       return(0);
```

Как правило, в качестве порогового значения для rates_total принимается значение периода индикатора, например для индикатора ADX, это rates_total <ExtADXPeriod.

```
//--- узнаем количество рассчитанных значений в индикаторе
  int calculated=BarsCalculated(handle);
      if(calculated<=0)
  {
      PrintFormat("BarsCalculated() вернул %d, код ошибки
%d",calculated,GetLastError());
      return(0);
  }
```

Если же в расчете буфера индикатора участвует хэндл другого индикатора, тогда проверяется количество рассчитанных данных для запрашиваемого индикатора.

С помощью функции BarsCalculated, которая принимает в качестве аргумента хэндл индикатора.

После проверки первоначальной загруженной истории для расчетов, вычисляется размер данных, которые необходимо рассчитать в этом вызове функции OnCalculate ().

В качестве примера, разберем блок кода функции OnCalculate, который приводится в справочнике, в разделе Технические индикаторы.

```
{
//--- количество копируемых значений из индикатора iMA
  int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
  int calculated=BarsCalculated(handle);
  if(calculated<=0)
    {
     PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
     return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений в индикаторе iMA
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось в истории)
  if( prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated+1 )
    {
     //--- если массив больше, чем значений в индикаторе на паре symbol/period, то копируем не все
     //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
     if(calculated>rates_total) values_to_copy=rates_total;
     else                       values_to_copy=calculated;
    }
  else
    {
     //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего вызова OnCalculate()
     //--- для расчета добавилось не более одного бара
     values_to_copy=(rates_total-prev_calculated)+1;
    }
. . . . . .
//--- запомним количество значений в индикаторе Moving Average
  bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
  return(rates_total);
 }
```

Здесь переменная values_to_copy – количество рассчитываемых значений в вызове функции OnCalculate ().

Переменная prev_calculated – сколько было обработано баров функцией OnCalculate () при предыдущем вызове.

Таким образом, при загрузке индикатора prev_calculated=0, а при каждом следующем поступлении нового тика prev_calculated= rates_total.

Переменная prev_calculated также обнуляется терминалом, если вдруг изменилось значение переменной rates_total.

Переменная bars_calculated – предыдущее количество рассчитанных данных для запрашиваемого индикатора, на основе которого рассчитывается данный индикатор.

Таким образом, первая проверка здесь, это prev_calculated==0 – индикатор только что загрузился или изменилась ценовая история.

calculated!=bars_calculated – изменилось количество рассчитанных данных для запрашиваемого индикатора.

rates_total> prev_calculated+1 – необходимо рассчитать индикатор для двух или более баров (значит, что-то изменилось в истории).

Последнее условие вступает в противоречие с утверждением справочника:

Если с момента последнего вызова функции OnCalculate () ценовые данные были изменены (подкачана более глубокая история или были заполнены пропуски истории), то значение входного параметра prev_calculated будет установлено в нулевое значение самим терминалом.

Если изменилась история, тогда сработает проверка prev_calculated==0 и проверка последнего условия будет излишней.

Теперь, если срабатывает первое или второе условие, тогда количество рассчитываемых значений – это размер входных таймсерий или количество рассчитанных данных для запрашиваемого индикатора, на основе которого рассчитывается данный индикатор, что будет из них меньше.

Если же первое или второе условие не срабатывают, тогда количество рассчитываемых значений:

values_to_copy= (rates_total-prev_calculated) +1;

Опять же, тут есть излишний код, так как, судя по справочнику, переменная prev_calculated может принимать значение либо 0, либо rates_total.

Поэтому, values_to_copy=1.

Таким образом, при поступлении нового тика, будет рассчитываться только одно значение индикатора для этого нового тика.

Рассмотрим другую реализацию вычисления размера данных, которые необходимо рассчитать в вызове функции OnCalculate ().

```
109 //--- we can copy not all data
110    int to_copy;
111    if(prev_calculated>rates_total || prev_calculated<0) to_copy=rates_total;
112    else
113      {
114       to_copy=rates_total-prev_calculated;
115       if(prev_calculated>0) to_copy++;
116      }
```

Для индикатора MACD это реализовано следующим образом.

Опять же, судя по справочнику, здесь будет работать только код:

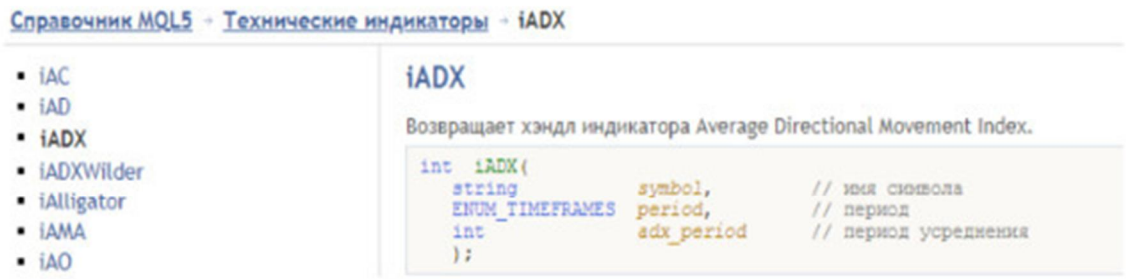
to_copy=rates_total-prev_calculated;

if (prev_calculated> 0) to_copy++;

Т.е. при загрузке индикатора to_copy=rates_total, а затем to_copy=1.

После вычисления размера данных, которые необходимо рассчитать в вызове функции OnCalculate (), производится их вычисление и заполнение ими буферов индикатора.

Если индикатор рассчитывается на основе хэндла другого индикатора, тогда производится копирование из буферов используемого индикатора в динамические массивы данного индикатора.



Вот как это реализовано в справочнике в примере к функции iADX, где используется индикатор ADX.

```
//--- заполняем часть массива ADXBuffer значениями из индикаторного буфера под индексом 0
   if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
     {
      //--- если копирование не удалось, сообщим код ошибки
      PrintFormat("Не удалось скопировать данные из индикатора iADX, код ошибки %d",GetLastError());
      //--- завершим с нулевым результатом - это означает, что индикатор будет считаться нерассчитанным
      return(false);
     }

//--- заполняем часть массива DI_plusBuffer значениями из индикаторного буфера под индексом 1
   if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
     {
      //--- если копирование не удалось, сообщим код ошибки
      PrintFormat("Не удалось скопировать данные из индикатора iADX, код ошибки %d",GetLastError());
      //--- завершим с нулевым результатом - это означает, что индикатор будет считаться нерассчитанным
      return(false);
     }

//--- заполняем часть массива DI_plusBuffer значениями из индикаторного буфера под индексом 2
   if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
     {
      //--- если копирование не удалось, сообщим код ошибки
      PrintFormat("Не удалось скопировать данные из индикатора iADX, код ошибки %d",GetLastError());
      //--- завершим с нулевым результатом - это означает, что индикатор будет считаться нерассчитанным
      return(false);
```

Здесь ind_handle – это хэндл индикатора ADX, второй параметр – индекс буфера используемого индикатора, из которого производится копирование, третий параметр – стартовая позиция, откуда начинается копирование.

Здесь мы помним, что копирование идет от конца к началу, и поэтому нулевая стартовая позиция – это самые свежие данные.

Четвертый параметр – это наш размер данных, которые необходимо рассчитать в вызове функции OnCalculate (), и последний параметр – это обычно динамический массив, привязанный к буферу индикатора, куда производится копирование.

Тут есть вопрос, как связать второй параметр функции CopyBuffer с индексом буфера используемого индикатора.

Это определяется вызовом функции SetIndexBuffer в используемом индикаторе.

```
SetIndexBuffer(0,ExtADXBuffer);
SetIndexBuffer(1,ExtPDIBuffer);
SetIndexBuffer(2,ExtNDIBuffer);
```

Например, для индикатора ADX.

Здесь нулевой индекс связан с буфером самого индикатора ADX, 1 индекс связан с буфером индикатора направленности +DI, 2 индекс связан с буфером индикатора направленности —DI.

Таким образом, для связывания второго параметра функции CopyBuffer с индексом буфера используемого индикатора, нужно знать код используемого индикатора.

```
for(int i=start; i<rates_total && !IsStopped(); i++)
   {
...
}
```

Также для заполнения буфера индикатора значениями, может использоваться цикл, например, цикл for.

Как это можно увидеть в коде индикаторов папки Examples редактора MetaEditor.

Здесь start – это стартовая позиция, с которой начинается заполнение буфера индикатора.

При значении prev_calculated=0, значение start это, как правило, 0, при значении prev_calculated= rates_total, значение start=prev_calculated-1.

Если же перед реализацией цикла с помощью функции ArraySetAsSeries поменять порядок доступа к массивам буферов индикатора и цен, тогда цикл примет следующий вид:

```
for(int i=start; i>=0; i--){
. . .

}
```

Где start=rates_total-1, если prev_calculated=0, и start=0, если prev_calculated= rates_total.

Пример создания индикатора

В качестве примера рассмотрим создание индикатора, который будет реализовывать форекс стратегию «Impulse keeper» (Ловец импульсов) и показывать на графике сигналы на покупку и продажу.



В данной стратегии применяются четыре индикатора:

– Экспоненциальная скользящая средняя с периодом 34 для цены High.

– Экспоненциальная скользящая средняя с периодом 34 для цены Low.

– Экспоненциальная скользящая средняя с периодом 125 для цены Close.

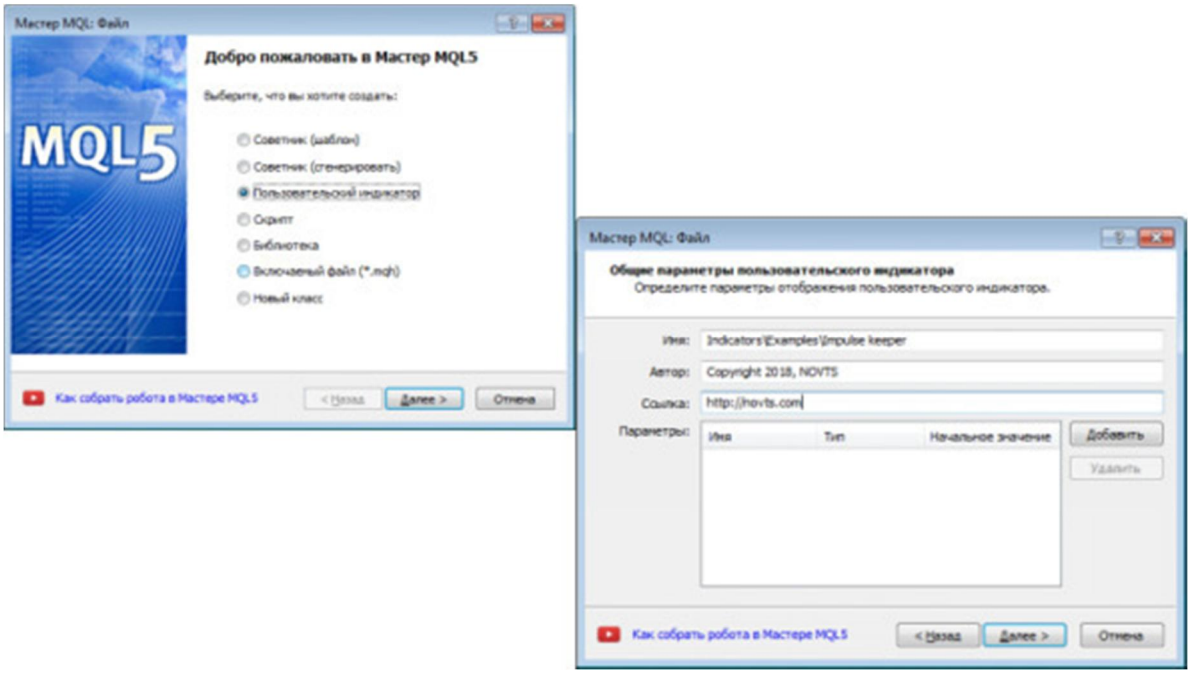
Parabolic SAR.

Сигналы на покупку и продажу в данной стратегии описываются следующим образом.

Сигнал на покупку: зеленая свеча закрывается выше EMA34 High и EMA34 Low, зеленая свеча выше EMA125 и Parabolic SAR.

Сигнал на продажу: красная свеча закрывается ниже EMA34 Low и EMA34 High, красная свеча ниже EMA125 и Parabolic SAR.

Давайте, реализуем эту стратегию в коде, который будет отображать на графике стрелки вверх и вниз сигналов на покупку и продажу.



Откроем MQL5 редактор и в меню Файл выберем Создать.

В диалоговом окне мастер MQL выберем Пользовательский индикатор и нажмем кнопку Далее.

Введем имя индикатора Impulse keeper, имя автора и ссылку и нажмем два раза Далее, а затем Готово.

```
//+------------------------------------------------------------------+
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#property indicator_chart_window
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---

//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

В результате мы получим код индикатора с пустыми функциями OnInit и OnCalculate.

Создание нашего индикатора начнем с определения его свойств.

```
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#property indicator_chart_window

#property indicator_buffers 8
#property indicator_plots 2
#property indicator_color1 clrGreen, clrBlack
#property indicator_type1 DRAW_COLOR_ARROW
#property indicator_color2 clrRed, clrBlack
#property indicator_type2 DRAW_COLOR_ARROW

double IKBuyBuffer[];
double ColorIKBuyBuffer[];
double IKSellBuffer[];
double ColorIKSellBuffer[];
double EMA34HBuffer[];
double EMA34LBuffer[];
double EMA125Buffer[];
double PSARBuffer[];

int EMA34HHandle;
int EMA34LHandle;
int EMA125Handle;
int PSARHandle;
```

Количество буферов индикатора определим 8.

2 буфера – данные и цвет, для сигналов на покупку.

2 буфера – данные и цвет, для сигналов на продажу.

И 4 буфера промежуточных вычислений для скопированных данных из индикаторов EMA34 Low, EMA34 High, EMA125 и Parabolic SAR:

#property indicator_buffers 8

Определим число графических построений – 2, одно построение для сигналов на покупку и другое построение для сигналов на продажу:

#property indicator_plots 2

Определим цвет и тип для обоих графических построений:

#property indicator_color1 clrGreen, clrBlack

#property indicator_type1 DRAW_COLOR_ARROW

#property indicator_color2 clrRed, clrBlack

#property indicator_type2 DRAW_COLOR_ARROW

Далее определим массивы буферов индикатора и хэндлы используемых индикаторов.

```mql5
int OnInit()
  {
PlotIndexSetInteger(0,PLOT_ARROW,233);
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,-10);

PlotIndexSetInteger(1,PLOT_ARROW,234);
PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
PlotIndexSetInteger(1,PLOT_ARROW_SHIFT,10);
//--- indicator buffers mapping
SetIndexBuffer(0,IKBuyBuffer,INDICATOR_DATA);
SetIndexBuffer(1,ColorIKBuyBuffer,INDICATOR_COLOR_INDEX);

SetIndexBuffer(2,IKSellBuffer,INDICATOR_DATA);
SetIndexBuffer(3,ColorIKSellBuffer,INDICATOR_COLOR_INDEX);

SetIndexBuffer(4,EMA34HBuffer,INDICATOR_CALCULATIONS);
SetIndexBuffer(5,EMA34LBuffer,INDICATOR_CALCULATIONS);
SetIndexBuffer(6,EMA125Buffer,INDICATOR_CALCULATIONS);
SetIndexBuffer(7,PSARBuffer,INDICATOR_CALCULATIONS);

EMA34HHandle=iMA(NULL,0,34,0,MODE_EMA,PRICE_HIGH);
EMA34LHandle=iMA(NULL,0,34,0,MODE_EMA,PRICE_LOW);
EMA125Handle=iMA(NULL,0,125,0,MODE_EMA,PRICE_CLOSE);
PSARHandle=iSAR(NULL,0,0.02, 0.2);

//---
   return(INIT_SUCCEEDED);
  }
```

В функции OnInit () для первого графического построения с индексом 0 определим тип стрелки – стрелка вверх, пустое значение и сдвиг, используя функции PlotIndexSetInteger и PlotIndexSetDouble.

Для второго графического построения с индексом 1 определим тип стрелки – стрелка вниз, пустое значение и сдвиг:

Свяжем массивы с буферами индикатора с помощью функции SetIndexBuffer.

И далее получим хэндлы используемых индикаторов, используя стандартные функции технических индикаторов iMA и iSAR.

```mql5
int bars_calculated=0;

  {
//---
   int values_to_copy;
   int start;
   int calculated=BarsCalculated(EMA34HHandle);
   if(calculated<=0)
     {
      return(0);
     }
   if(prev_calculated==0 || calculated!=bars_calculated)
     {
      start=1;
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
     start=rates_total-1;
     values_to_copy=1;
     }

bars_calculated=calculated;
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

В функции OnCalculate () произведем проверку размера доступной истории для расчета используемых индикаторов calculated, определим количество копируемых значений используемых индикаторов values_to_copy и определим стартовую позицию расчета индикатора start.

И переменную bars_calculated определим как глобальную int bars_calculated=0; в свойствах индикатора.

```
99 if(!FillArrayFromMABuffer(EMA34HBuffer,0,EMA34HHandle,values_to_copy)) return(0);
100
101 if(!FillArrayFromMABuffer(EMA34LBuffer,0,EMA34LHandle,values_to_copy)) return(0);
102
103 if(!FillArrayFromMABuffer(EMA125Buffer,0,EMA125Handle,values_to_copy)) return(0);
104
105 if(!FillArrayFromPSARBuffer(PSARBuffer,PSARHandle,values_to_copy)) return(0);
```

Далее произведем копирование из буферов используемых индикаторов в массивы буферов нашего индикатора.

Здесь FillArrayFromMABuffer и FillArrayFromPSARBuffer – пользовательские функции, определенные вне функции OnCalculate ().

```mql5
//+------------------------------------------------------------------+
bool FillArrayFromPSARBuffer(
double &sar_buffer[],   // индикаторный буфер значений Parabolic SAR
int ind_handle,         // хэндл индикатора iSAR
int amount              // количество копируемых значений
                        )
  {
   ResetLastError();
   if(CopyBuffer(ind_handle,0,0,amount,sar_buffer)<0)
     {
          return(false);
     }
   return(true);
  }
//+------------------------------------------------------------------+
bool FillArrayFromMABuffer(
double &values[],    // индикаторный буфер значений Moving Average
int shift,           // смещение
int ind_handle,      // хэндл индикатора iMA
int amount           // количество копируемых значений
                        )
  {
   ResetLastError();
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
         return(false);
     }
   return(true);
  }
```

Функция FillArrayFromPSARBuffer отвечает за копирование данных индикатора Parabolic SAR в указанный массив, используя функцию CopyBuffer.

А функция FillArrayFromMABuffer отвечает за копирование данных индикатора Moving Average в указанный массив.

```mql5
for(int i=start;i<rates_total && !IsStopped();i++)
    {
    IKBuyBuffer[i-1]=0;
    ColorIKBuyBuffer[i-1]=1;

    IKSellBuffer[i-1]=0;
    ColorIKSellBuffer[i-1]=1;

if(close[i-1]>open[i-1]&&close[i-1]>EMA34HBuffer[i-1]&&close[i-1]>EMA34LBuffer[i-1]
&&low[i-1]>EMA125Buffer[i-1]&&low[i-1]>PSARBuffer[i-1]&&EMA125Buffer[i-1]<EMA34LBuffer[i-1]
&&EMA125Buffer[i-1]<EMA34HBuffer[i-1]){
    IKBuyBuffer[i-1]=high[i-1];
    ColorIKBuyBuffer[i-1]=0;
    }

if(close[i-1]<open[i-1]&&close[i-1]<EMA34HBuffer[i-1]&&close[i-1]<EMA34LBuffer[i-1]
&&high[i-1]<EMA125Buffer[i-1]&&high[i-1]<PSARBuffer[i-1]&&EMA125Buffer[i-1]>EMA34LBuffer[i-1]
&&EMA125Buffer[i-1]>EMA34HBuffer[i-1]){
    IKSellBuffer[i-1]=low[i-1];
    ColorIKSellBuffer[i-1]=0;
    }
    }
```

Далее в функции OnCalculate () заполним буферы индикатора данными и цветом.

Здесь мы рассчитываем индикатор на предыдущем баре, так как на текущем баре цена close – это текущая цена тика.

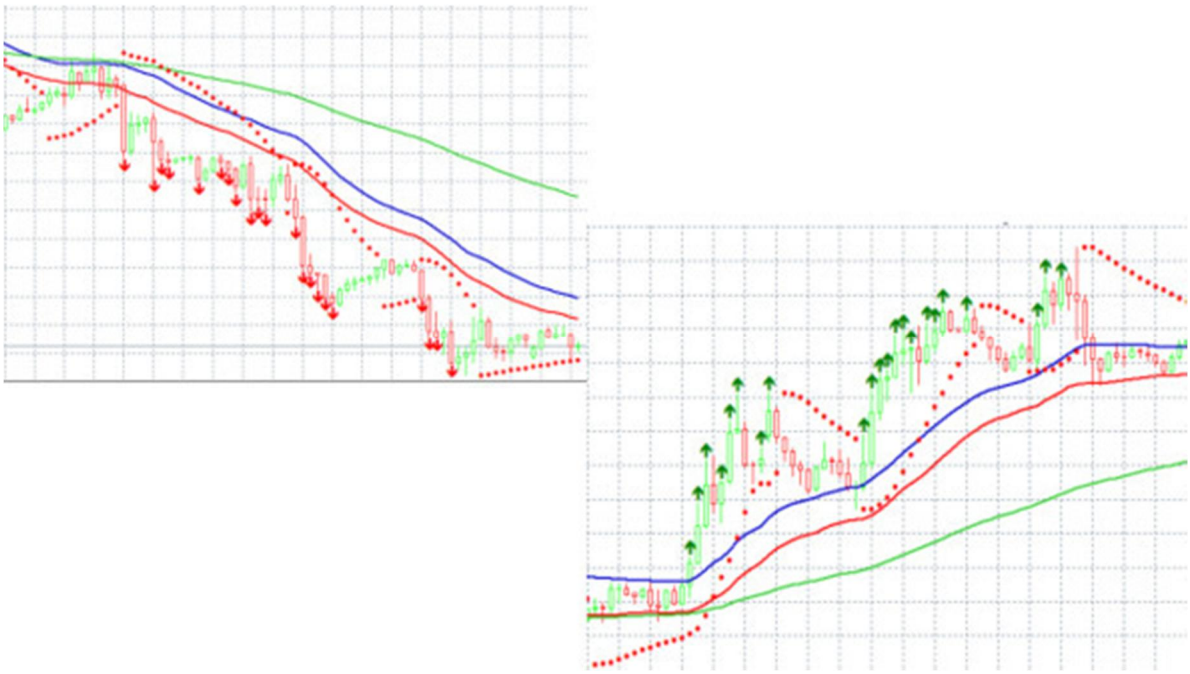
И здесь в цикле for проходя по ценовой истории, мы сначала устанавливаем значение нашего индикатора в 0 и его цвет как черный.

При этом, так как мы с помощью функции PlotIndexSetDouble установили нулевое значение индикатора как значение, которое не будет отрисовываться, значения индикатора черного цвета не будут отображаться.

Затем мы проверяем, соответствуют ли условия, согласно нашей стратегии, покупке или продаже финансового инструмента.

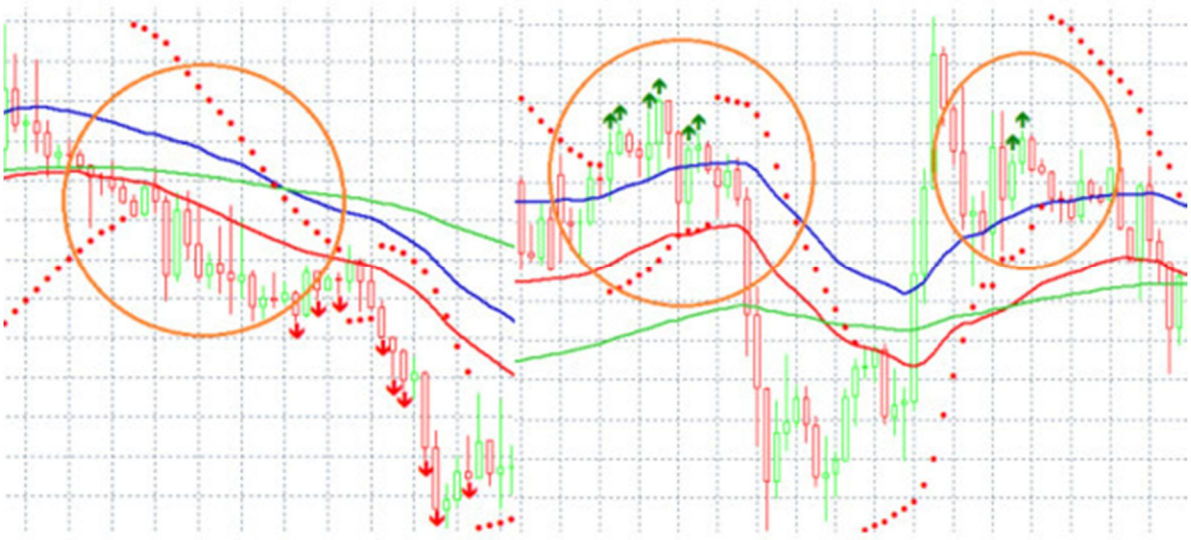
И если условия соответствуют покупке, тогда мы устанавливаем значение индикатора как цену high бара, а его цвет как зеленый.

Если же условия соответствуют продаже, тогда мы устанавливаем значение индикатора как цену low бара, а его цвет как красный.



Откомпилируем код и присоединим индикатор к графику.

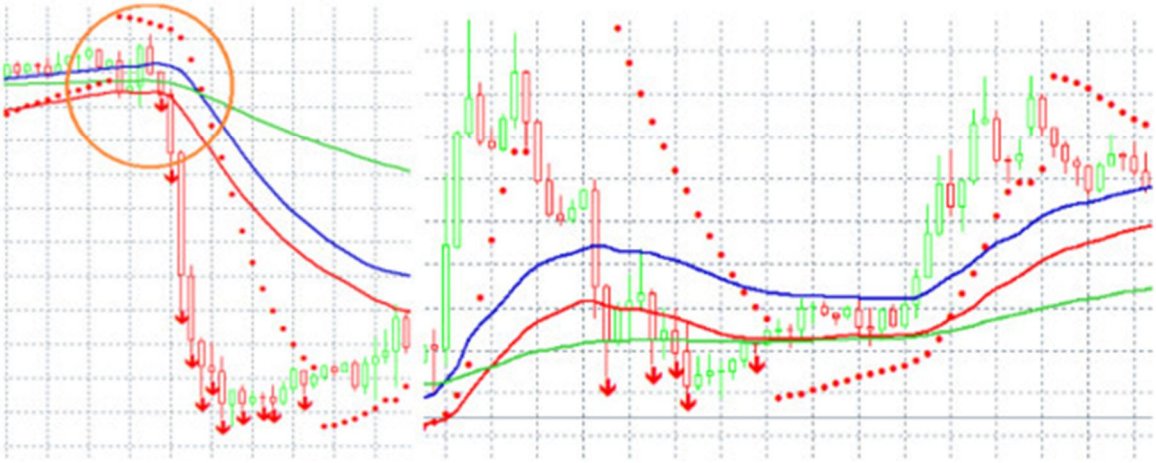
Мы увидим, что, в общем и целом, индикатор дает верные сигналы на продажу и покупку, хотя в некоторых случаях он запаздывает и дает ложные сигналы.



Как мы видим, происходит это из-за трендовой линии EMA125.

Попробуем отвязать ее от текущего периода и попробуем определять тренд, скажем по дневному графику.

```
57 EMA34HHandle=iMA(NULL,0,34,0,MODE_EMA,PRICE_HIGH);
58 EMA34LHandle=iMA(NULL,0,34,0,MODE_EMA,PRICE_LOW);
59 //EMA125Handle=iMA(NULL,0,125,0,MODE_EMA,PRICE_CLOSE);
60 PSARHandle=iSAR(NULL,0,0.02, 0.2);
61
62 EMA125Handle=iMA(NULL,PERIOD_D1,125,0,MODE_EMA,PRICE_CLOSE);
```



При этом запаздывание, конечно, сократится, но количество ложных сигналов увеличится.

Видимо для улучшения данной стратегии, нужно привлекать дополнительные индикаторы.

Попробуем сделать этот же самый индикатор, но не с помощью графических построений, а помещая графические объекты на график символа.

В свойствах индикатора теперь не нужно объявлять буферы данных и цвета индикатора и графические серии для них.

```
//+------------------------------------------+
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#property indicator_chart_window

#property indicator_buffers 4

double EMA34HBuffer[];
double EMA34LBuffer[];
double EMA125Buffer[];
double PSARBuffer[];

int EMA34HHandle;
int EMA34LHandle;
int EMA125Handle;
int PSARHandle;

int    bars_calculated=0;
```

Оставим только буферы индикатора для промежуточных расчетов и хэндлы используемых индикаторов.

```
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,EMA34HBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(1,EMA34LBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(2,EMA125Buffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(3,PSARBuffer,INDICATOR_CALCULATIONS);

   EMA34HHandle=iMA(NULL,0,34,0,MODE_EMA,PRICE_HIGH);
   EMA34LHandle=iMA(NULL,0,34,0,MODE_EMA,PRICE_LOW);
   EMA125Handle=iMA(NULL,0,125,0,MODE_EMA,PRICE_CLOSE);
   PSARHandle=iSAR(NULL,0,0.02, 0.2);
//---
   return(INIT_SUCCEEDED);
  }
```

В функции OnInit () соответственно оставим только привязку массивов к буферам промежуточных расчетов и получение хэндлов используемых индикаторов.

```
   for(int i=start;i<rates_total && !IsStopped();i++)
     {
if(close[i-1]>open[i-1]&&close[i-1]>EMA34HBuffer[i-1]&&close[i-1]>EMA34LBuffer[i-1]
&&low[i-1]>EMA128Buffer[i-1]&&low[i-1]>PSARBuffer[i-1]&&EMA128Buffer[i-1]<EMA34LBuffer[i-1]
&&EMA128Buffer[i-1]<EMA34HBuffer[i-1]){

      if(!ObjectCreate(0,"Buy"+i,OBJ_ARROW,0,time[i-1],high[i-1]))
      {
       return(false);
      }
       ObjectSetInteger(0,"Buy"+i,OBJPROP_COLOR,clrGreen);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_ARROWCODE,233);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_WIDTH,2);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_ANCHOR,ANCHOR_UPPER);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_HIDDEN,true);
       ObjectSetString(0,"Buy"+i,OBJPROP_TOOLTIP,""+close[i-1]);
      }
if(close[i-1]<open[i-1]&&close[i-1]<EMA34HBuffer[i-1]&&close[i-1]<EMA34LBuffer[i-1]
&&high[i-1]<EMA128Buffer[i-1]&&high[i-1]<PSARBuffer[i-1]&&EMA128Buffer[i-1]>EMA34LBuffer[i-1]
&&EMA128Buffer[i-1]>EMA34HBuffer[i-1]){

      if(!ObjectCreate(0,"Sell"+i,OBJ_ARROW,0,time[i-1],low[i-1]))
      {
       return(false);
      }
       ObjectSetInteger(0,"Sell"+i,OBJPROP_COLOR,clrRed);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_ARROWCODE,234);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_WIDTH,2);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_ANCHOR,ANCHOR_LOWER);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_HIDDEN,true);
       ObjectSetString(0,"Sell"+i,OBJPROP_TOOLTIP,""+close[i-1]);
      }
      }
```

В функции OnCalculate определим создание объектов на графике символа.

Здесь функцией ObjectCreate создаются объекты стрелки, привязанные ко времени и максимальной или минимальной цене.

И создаются два разных объекта стрелки в зависимости от того, соответствуют ли условия покупке или продаже финансового инструмента.

После создания графического объекта стрелки, функцией ObjectSetInteger со свойством OBJPROP_COLOR определяется цвет стрелки.

Функцией ObjectSetInteger со свойством OBJPROP_ARROWCODE определяется направление стрелки вверх или вниз.

Функцией ObjectSetInteger со свойством OBJPROP_WIDTH определяется размер объекта.

Функцией ObjectSetInteger со свойством OBJPROP_ANCHOR определяется привязка к цене сверху или снизу по центру.

Функцией ObjectSetInteger со свойством OBJPROP_HIDDEN – true определяется отсутствие созданных объектов в списке объектов графика символа.

Функцией ObjectSetString со свойством OBJPROP_TOOLTIP определяется содержание всплывающей подсказки при наведении указателя на объект.

```
119 void  OnDeinit(const int reason){
120 ObjectsDeleteAll(0,-1,-1);
121 }
```

Теперь, в функции OnDeinit уберем все добавленные графические объекты, используя функцию ObjectsDeleteAll.

И более подробно о создании объектов на графике символа мы еще поговорим позже.



Кстати, мы использовали функцию ObjectSetInteger со свойством OBJPROP_HIDDEN – true, чтобы не засорять список объектов графика символа нашими созданными объектами стрелки.

Графические объекты

Как уже было показано ранее, мы можем рисовать на графике символа не только диаграммы индикатора, но и добавлять различные графические объекты с помощью функции ObjectCreate.

ObjectCreate

Создает объект с указанным именем, типом и начальными координатами в указанном подокне графика. При создании можно указать до 30 координат.

```
bool  ObjectCreate(
   long          chart_id,      // идентификатор графика
   string        name,          // имя объекта
   ENUM_OBJECT   type,          // тип объекта
   int           sub_window,    // индекс окна
   datetime      time1,         // время первой точки привязки
   double        price1,        // цена первой точки привязки
   ...
   datetime      timeN=0,       // время N-ой точки привязки
   double        priceN=0,      // цена N-ой точки привязки
   ...
   datetime      time30=0,      // время 30-й точки привязки
   double        price30=0      // цена 30-точки привязки
   );
```

Здесь параметр sub_window это индекс главного окна графика символа со значением 0 или индекс подокна другого индикатора, присоединенного к графику символа.

```
 99       if(!ObjectCreate(0,"Buy1"+i,OBJ_ARROW,1,time[i-1],high[i-1]))
100      {
101       return(false);
102      }
103
104       ObjectSetInteger(0,"Buy1"+i,OBJPROP_COLOR,clrGreen);
105       ObjectSetInteger(0,"Buy1"+i,OBJPROP_ARROWCODE,233);
106       ObjectSetInteger(0,"Buy1"+i,OBJPROP_WIDTH,2);
107       ObjectSetInteger(0,"Buy1"+i,OBJPROP_ANCHOR,ANCHOR_UPPER);
108       ObjectSetInteger(0,"Buy1"+i,OBJPROP_HIDDEN,true);
109       ObjectSetString(0,"Buy1"+i,OBJPROP_TOOLTIP,close[i-1]);

128       if(!ObjectCreate(0,"Sell1"+i,OBJ_ARROW,1,time[i-1],low[i-1]))
129      {
130       return(false);
131      }
132
133       ObjectSetInteger(0,"Sell1"+i,OBJPROP_COLOR,clrRed);
134       ObjectSetInteger(0,"Sell1"+i,OBJPROP_ARROWCODE,234);
135       ObjectSetInteger(0,"Sell1"+i,OBJPROP_WIDTH,2);
136       ObjectSetInteger(0,"Sell1"+i,OBJPROP_ANCHOR,ANCHOR_LOWER);
137       ObjectSetInteger(0,"Sell1"+i,OBJPROP_HIDDEN,true);
138       ObjectSetString(0,"Sell1"+i,OBJPROP_TOOLTIP,close[i-1]);
```

Например, если в предыдущем примере с пользовательским индикатором Impulse Keeper, мы изменим код, добавив объекты стрелки в подокно с индексом 1,

И присоединим к графику символа, скажем, индикатор ADX, мы увидим следующее:



Нумерация подокон идет сверху вниз в порядке отображения.

Третий параметр функции ObjectCreate – тип отображаемого объекта задается перечислением ENUM_OBJECT, которое можно посмотреть в справочнике.

Типы объектов

При создании графического объекта функцией ObjectCreate() необходимо указать тип создаваемого объекта, который может принимать одно из значений перечисления ENUM_OBJECT. Дальнейшие уточнения свойств созданного объекта возможно с помощью функций по работе с графическими объектами.

ENUM_OBJECT

Идентификатор		Описание
OBJ_VLINE		Вертикальная линия
OBJ_HLINE		Горизонтальная линия
OBJ_TREND		Трендовая линия
OBJ_TRENDBYANGLE		Трендовая линия по углу
OBJ_CYCLES		Циклические линии
OBJ_ARROWED_LINE		Объект "Линия со стрелкой"
OBJ_CHANNEL		Равноудаленный канал
OBJ_STDDEVCHANNEL		Канал стандартного отклонения

После добавления графических объектов, не забываем их удалять в функции обратного вызова OnDeinit, используя функцию ObjectDelete:

```
bool  ObjectDelete(
  long    chart_id,    // chart identifier
  string  name         // object name
  );

int  ObjectsDeleteAll(
  long  chart_id,          // chart identifier
  int   sub_window=-1,     // window index
  int   type=-1            // object type
  );
```

Или используя функцию ObjectsDeleteAll.

Где sub_window=-1 означает все подокна графика, включая главное окно.



Помимо вышеупомянутых функций ObjectCreate, ObjectDelete и ObjectsDeleteAll, MQL5 предлагает целый набор функций для работы с графическими объектами: ObjectName, ObjectFind, и другие.

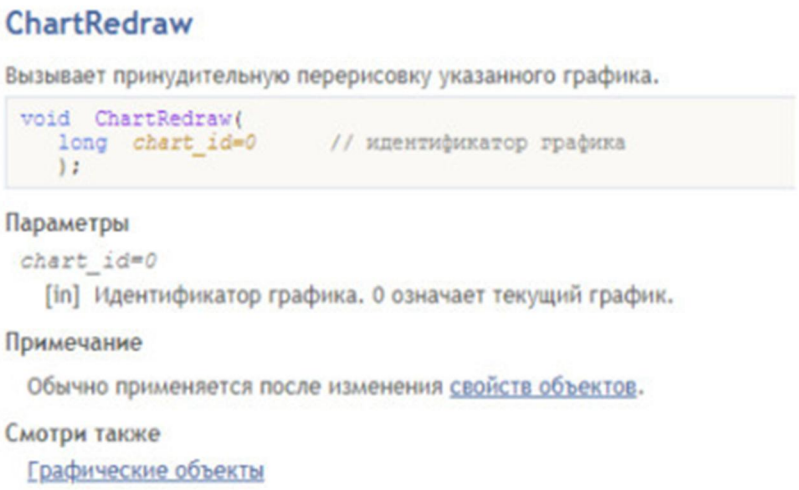
Функции ObjectName, ObjectFind, ObjectGetTimeByValue, ObjectGetValueByTime, ObjectsTotal, ObjectGetDouble, ObjectGetInteger, ObjectGetString, TextGetSize – это функции, возвращающие информацию.

Функции ObjectSetDouble, ObjectSetInteger, ObjectSetString, TextSetFont – это функции устанавливающие свойства объекта.

Функция ObjectMove перемещает объект в окне.

Функция TextOut выводит текст в пиксельный массив для отображения объектом OBJ_BITMAP_LABEL или OBJ_BITMAP.

После добавления графических объектов рекомендуется принудительно перерисовать график символа с помощью функции ChartRedraw.



ChartRedraw

Вызывает принудительную перерисовку указанного графика.

```
void  ChartRedraw(
   long  chart_id=0      // идентификатор графика
   );
```

Параметры

chart_id=0

[in] Идентификатор графика. 0 означает текущий график.

Примечание

Обычно применяется после изменения свойств объектов.

Смотри также

Графические объекты

Надо отметить, что функция ObjectCreate позволяет создавать программным способом те графические объекты, которые вы можете вручную нарисовать на графике символа, пользуясь панелью инструментов клиентского терминала.

ObjectSetDouble

Задает значение соответствующего свойства объекта. Свойство объекта должно быть типа double.

Установка значения свойства, не имеющего модификатора

```
bool  ObjectSetDouble(
   long                     chart_id,        // идентификатор графика
   string                   name,            // имя
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,     // свойство
   double                   prop_value       // значение
   );
```

Установка значения свойства с указанием модификатора

```
bool  ObjectSetDouble(
   long                     chart_id,        // идентификатор графика
   string                   name,            // имя
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,     // свойство
   int                      prop_modifier,   // модификатор
   double                   prop_value       // значение
   );
```

С помощью функции ObjectSetDouble устанавливаются такие свойства графического объекта, как OBJPROP_PRICE – изменение параметра price функции ObjectCreate,

OBJPROP_LEVELVALUE – определение уровней для таких объектов, как инструменты Фиббоначи и Вилы Эндрюса,

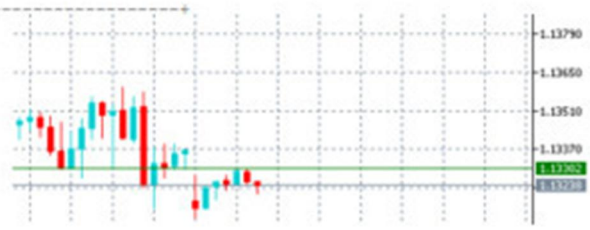
OBJPROP_SCALE – определение масштаба для таких объектов, как инструменты Ганна и Дуги Фибоначчи,

OBJPROP_ANGLE – определение угла объекта, т.е. возможность повернуть объект, который изначально не имеет жесткой привязки, например, повернуть текст,

OBJPROP_DEVIATION – определение отклонения для объекта Канал стандартного отклонения.

Как пример использования OBJPROP_PRICE:

```mql5
ArraySetAsSeries(time, true);
ArraySetAsSeries(high, true);
ArraySetAsSeries(low, true);
ArraySetAsSeries(close, true);
ObjectDelete(0,"Price");
    if(!ObjectCreate(0,"Price",OBJ_HLINE,0,time[1],close[1]))
     {
      return(false);
     }
      ObjectSetInteger(0,"Price",OBJPROP_COLOR,clrGreen);
      ObjectSetInteger(0,"Price",OBJPROP_WIDTH,1);
      ObjectSetString(0,"Price",OBJPROP_TOOLTIP,close[1]);
      if (open[1]>close[1])
      ObjectSetDouble(0,"Price",OBJPROP_PRICE,low[1]);
      if (open[1]<close[1])
      ObjectSetDouble(0,"Price",OBJPROP_PRICE,high[1]);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
void  OnDeinit(const int reason){
ObjectsDeleteAll(0,-1,-1);
}
```



В этом коде создается горизонтальный уровень, показывающий минимальную или максимальную цену предыдущего бара, в зависимости от того, является ли этот бар бычьим или медвежьим.

Пример использования OBJPROP_ANGLE:

```mql5
ArraySetAsSeries(time, true);
ArraySetAsSeries(high, true);
ArraySetAsSeries(low, true);
ArraySetAsSeries(close, true);
ObjectDelete(0,"Line");
ObjectDelete(0,"Price");
    if(!ObjectCreate(0,"Line",OBJ_VLINE,0,time[1],close[1]))
     {
      return(false);
     }
      ObjectSetInteger(0,"Line",OBJPROP_COLOR,clrBlue);
      ObjectSetInteger(0,"Line",OBJPROP_WIDTH,1);
      ObjectSetString(0,"Line",OBJPROP_TOOLTIP,close[1]);

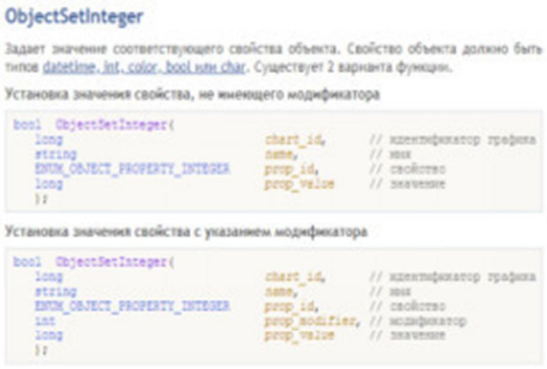
      if(!ObjectCreate(0,"Price",OBJ_TEXT,0,time[3],high[1]))
     {
      return(false);
     }
      ObjectSetString(0,"Price",OBJPROP_TEXT,close[1]);
      ObjectSetInteger(0,"Price",OBJPROP_COLOR,clrBlack);
      ObjectSetDouble(0,"Price",OBJPROP_ANGLE,90);
      ObjectSetString(0,"Price",OBJPROP_TOOLTIP,close[1]);

//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
void  OnDeinit(const int reason){
ObjectsDeleteAll(0,-1,-1);
}
```



Этот код создает вертикальную линию с подписью цены закрытия предыдущего бара.

С помощью функции ObjectSetInteger устанавливаются такие свойства графического объекта, как цвет, стиль, размер и др.



ObjectSetInteger

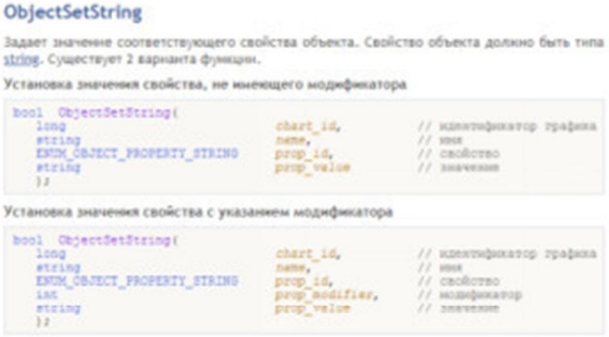
Задает значение соответствующего свойства объекта. Свойство объекта должно быть типов datetime, int, color, bool или char. Существует 2 варианта функции.

Установка значения свойства, не имеющего модификатора

```
bool  ObjectSetInteger(
   long                    chart_id,      // идентификатор графика
   string                  name,          // имя
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id, // свойство
   long                    prop_value     // значение
   );
```

Установка значения свойства с указанием модификатора

```
bool  ObjectSetInteger(
   long                    chart_id,      // идентификатор графика
   string                  name,          // имя
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id, // свойство
   int                     prop_modifier, // модификатор
   long                    prop_value     // значение
   );
```



ObjectSetString

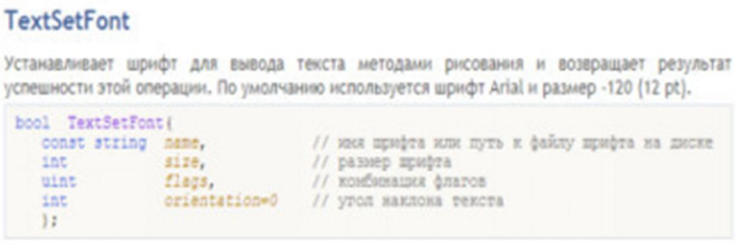
Задает значение соответствующего свойства объекта. Свойство объекта должно быть типа string. Существует 2 варианта функции.

Установка значения свойства, не имеющего модификатора

```
bool  ObjectSetString(
   long                    chart_id,      // идентификатор графика
   string                  name,          // имя
   ENUM_OBJECT_PROPERTY_STRING   prop_id, // свойство
   string                  prop_value     // значение
   );
```

Установка значения свойства с указанием модификатора

```
bool  ObjectSetString(
   long                    chart_id,      // идентификатор графика
   string                  name,          // имя
   ENUM_OBJECT_PROPERTY_STRING   prop_id, // свойство
   int                     prop_modifier, // модификатор
   string                  prop_value     // значение
   );
```



TextSetFont

Устанавливает шрифт для вывода текста методами рисования и возвращает результат успешности этой операции. По умолчанию используется шрифт Arial и размер -120 (12 pt).

```
bool  TextSetFont(
   const string  name,           // имя шрифта или путь к файлу шрифта на диске
   int           size,           // размер шрифта
   uint          flags,          // комбинация флагов
   int           orientation=0   // угол наклона текста
   );
```

С помощью функции ObjectSetString можно изменить имя объекта, при этом объект со старым именем будет удален и будет создан объект с новым именем, установить текст для таких объектов, как текст, кнопка, метка, поле ввода, событие, установить текст всплывающей подсказки для объекта, описание уровня для объектов, имеющих уровни, шрифт, имя BMP-файла для объекта «Графическая метка» и «Рисунок», символ для объекта «График».

Функция TextSetFont позволяет установить тип шрифта текста, его размер, стиль и угол наклона для объектов, содержащих текст.

```
uint ExtImg[10000];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
ObjectCreate(0,"Image",OBJ_BITMAP_LABEL,0,0,0);
ObjectSetString(0,"Image",OBJPROP_BMPFILE,"::IMG");
ArrayFill(ExtImg,0,10000,0xffffff);
TextOut("Text",10,10,TA_LEFT|TA_TOP,ExtImg,100,100,0x000000,COLOR_FORMAT_XRGB_NOALPHA);
ResourceCreate("::IMG",ExtImg,100,100,0,0,0,COLOR_FORMAT_XRGB_NOALPHA);
ChartRedraw();

//---
   return(INIT_SUCCEEDED);
  }
```

Как уже было сказано, функция TextOut позволяет скомбинировать текст и изображение.

Например, показанный на слайде код выводит текст в изображение, залитое одним цветом.

Здесь ExtImg это пиксельный массив, представляющий изображение 100x100 пикселей.

Функция ObjectCreate создает объект «Графическая метка» OBJ_BITMAP_LABEL, а функция ObjectSetString устанавливает для этого объекта файл изображения с именем::IMG. По поводу знака «::» справочник говорит следующее:

Для использования своего ресурса в коде нужно перед именем ресурса добавлять специальный признак "::».

Функция ArrayFill заполняет пиксельный массив пикселями белого цвета.

Функция TextOut выводит в пиксельный массив слово «Text».

Функция ResourceCreate создает из пиксельного массива ресурс с именем::IMG.

В итоге на белом фоне отображается надпись «Text».

```mql5
#resource "\\Images\\image.bmp"
uint ExtImg[10000];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
ObjectCreate(0,"Image",OBJ_BITMAP_LABEL,0,0,0);
ObjectSetString(0,"Image",OBJPROP_BMPFILE,"::IMG");
uint width=100;
uint height=100;
ResourceReadImage("::Images\\image.bmp",ExtImg,width,height);
TextOut("Text",10,10,TA_LEFT|TA_TOP,ExtImg,100,100,0xffffff,COLOR_FORMAT_ARGB_NORMALIZE);
ResourceCreate("::IMG",ExtImg,100,100,0,0,0,COLOR_FORMAT_XRGB_NOALPHA);
ChartRedraw();
```

Также можно вывести текст на готовое изображение.

Здесь функция ResourceReadImage считывает существующее изображение из папки Images окна Navigator редактора MQL5 в пиксельный массив::IMG, связанный с объектом «Графическая метка», а функция TextOut выводит в пиксельный массив слово «Text».

```mql5
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   //---
ArraySetAsSeries(time, true);
ArraySetAsSeries(high, true);
ArraySetAsSeries(low, true);
ArraySetAsSeries(close, true);
ObjectDelete(0,"Image");
ObjectCreate(0,"Image",OBJ_BITMAP,0,time[1],close[1]);
ObjectSetString(0,"Image",OBJPROP_BMPFILE,"::IMG");
uint width=100;
uint height=100;
ResourceReadImage("::Images\\image.bmp",ExtImg,width,height);
TextOut("Text",10,10,TA_LEFT|TA_TOP,ExtImg,100,100,0xffffff,COLOR_FORMAT_ARGB_NORMALIZE);
ResourceCreate("::IMG",ExtImg,100,100,0,0,0,COLOR_FORMAT_XRGB_NOALPHA);
ChartRedraw();
```

То же самое можно проделать и с объектом «Рисунок» OBJ_BITMAP.

Здесь мы в функции OnCalculate создаем объект «Рисунок» с помощью функции ObjectCreate, которая прикрепляет рисунок к цене закрытия последнего бара.

И далее мы заполняем связанный с этим объектом файл, текстом и изображением.

В качестве примера использования графических объектов, рассмотрим создание индикатора, который выводит в небольшое окно на графике символа тот же график, но с другим временным периодом.

Для этого используем графический объект OBJ_CHART.

```
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#property indicator_chart_window

input string           InpSymbol="EURUSD";          // Символ
input ENUM_TIMEFRAMES  InpPeriod=PERIOD_CURRENT;    // Период
```

В качестве входных параметров индикатора используем символ графика и его период.

```
17 int OnInit()
18   {
19   if(!ObjectCreate(0,"Chart",OBJ_CHART,0,0,0))
20      {
21            return(false);
22      }
23 ObjectSetInteger(0,"Chart",OBJPROP_XDISTANCE,10);
24 ObjectSetInteger(0,"Chart",OBJPROP_YDISTANCE,20);
25 ObjectSetInteger(0,"Chart",OBJPROP_XSIZE,300);
26 ObjectSetInteger(0,"Chart",OBJPROP_YSIZE,200);
27 ObjectSetString(0,"Chart",OBJPROP_SYMBOL,InpSymbol);
28 ObjectSetInteger(0,"Chart",OBJPROP_PERIOD,InpPeriod);
29 ObjectSetInteger(0,"Chart",OBJPROP_DATE_SCALE,true);
30 ObjectSetInteger(0,"Chart",OBJPROP_WIDTH,1);
31 ObjectSetInteger(0,"Chart",OBJPROP_PRICE_SCALE,true);
32 ObjectSetInteger(0,"Chart",OBJPROP_SELECTABLE,true);
33 ObjectSetInteger(0,"Chart",OBJPROP_SELECTED,true);
34 ObjectSetInteger(0,"Chart",OBJPROP_COLOR,clrBlue);
```

В функции OnInit создадим графический объект График OBJ_CHART.

По умолчанию точка привязки этого объекта – левый верхний угол графика.

Определим отступ точки привязки объекта, его размеры, символ и период графика, отображение шкалы времени, размер точки привязки, с помощью которой можно перемещать объект, отображение ценовой шкалы, режим перемещения мышкой, цвет рамки графика.

```
long chartId=ObjectGetInteger(0,"Chart",OBJPROP_CHART_ID);
```

https://www.mql5.com/ru/docs/chart_operations

Операции с графиками

Функции для установки свойств графика (ChartSetInteger, ChartSetDouble, ChartSetString) являются асинхронными и служат для отправки графику команд на изменение. При успешном выполнении этих функций команда попадает в общую очередь событий графика. Изменение свойств графика производится в процессе обработки очереди событий данного графика.

По этой причине не следует ожидать немедленного обновления графика после вызова асинхронных функций. Для принудительного обновления внешнего вида и свойств графика используйте функцию ChartRedraw().

Функция	Действие
ChartApplyTemplate	Применяет к указанному графику шаблон из указанного файла
ChartSaveTemplate	Сохраняет текущие настройки графика в шаблон с указанным именем
ChartWindowFind	Возвращает номер подокна, в котором находится индикатор
ChartTimePriceToXY	Преобразует координаты графика из представления время/цена в координаты по оси X и Y
ChartXYToTimePrice	Преобразует координаты X и Y графика в значения время и цена
ChartOpen	Открывает новый график с указанным символом и периодом

https://www.mql5.com/ru/docs/constants/chartconstants/enum_chart_property

Свойства графиков

Идентификаторы семейства перечислений ENUM_CHART_PROPERTY используются в качестве параметров функций для работы с графиками. Аббревиатура r/o в столбце "Тип свойства" означает, что данное свойство предназначено только для чтения и не может быть изменено. Аббревиатура w/o в столбце "Тип свойства" означает, что данное свойство предназначено только для записи и не может быть получено. При обращении к некоторым свойствам необходимо указывать дополнительный параметр-модификатор (modifier), который служит для указания номера подокна графика. 0 означает главное окно.

Функции, устанавливающие свойства графика, фактически служат для отправки ему команд на изменение. При успешном выполнении этих функций команда попадает в общую очередь событий графика. Изменение графика производится в процессе обработки очереди событий данного графика.

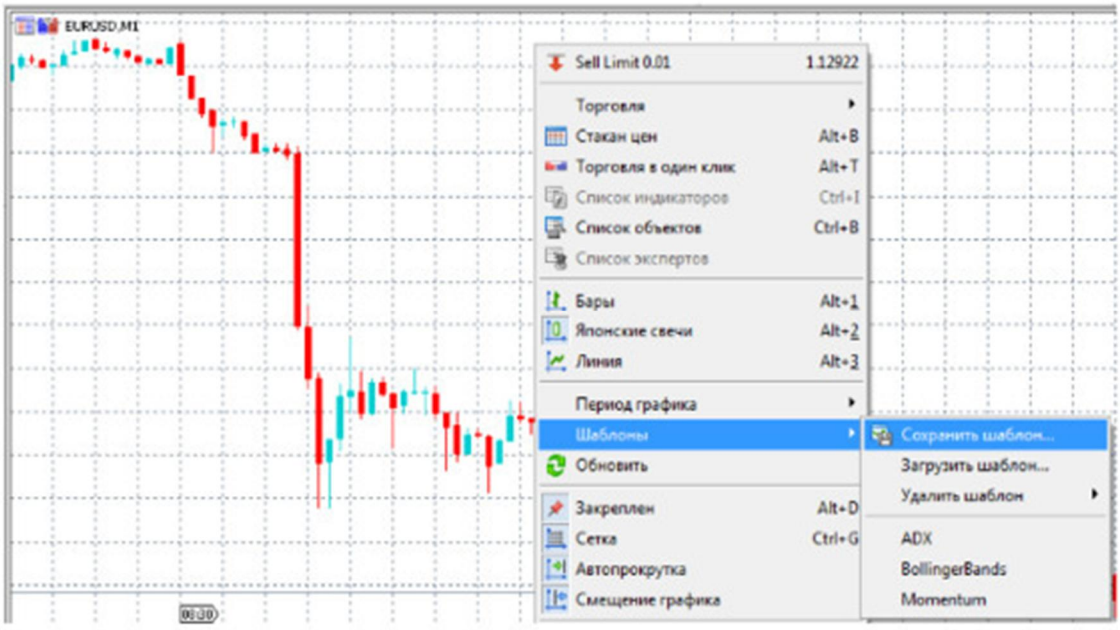
По этой причине не следует ожидать немедленного визуального обновления графика после вызова данных функций. В общем случае обновление графика производится терминалом автоматически по событиям изменения - поступление новой котировки, изменения размера окна графика и т.д. Для принудительного обновления внешнего вида графика используйте команду на перерисовку графика ChartRedraw().

Для функций ChartSetInteger() и ChartGetInteger()

ENUM_CHART_PROPERTY_INTEGER

Идентификатор	Описание	Тип свойства

С помощью свойства объектов OBJPROP_CHART_ID функции ObjectGetInteger получим идентификатор графика, используя который мы теперь можем применять функции работы с графиками и свойства графиков.



Откроем наш график символа, к которому мы хотим присоединить индикатор, и нажав правой кнопкой мышки, выберем пункт в контекстном меню Шаблоны и Сохранить шаблон.

```
int OnInit()
  {
  if(!ObjectCreate(0,"Chart",OBJ_CHART,0,0,0))
     {
           return(false);
     }
ObjectSetInteger(0,"Chart",OBJPROP_XDISTANCE,10);
ObjectSetInteger(0,"Chart",OBJPROP_YDISTANCE,20);
ObjectSetInteger(0,"Chart",OBJPROP_XSIZE,300);
ObjectSetInteger(0,"Chart",OBJPROP_YSIZE,200);
ObjectSetString(0,"Chart",OBJPROP_SYMBOL,InpSymbol);
ObjectSetInteger(0,"Chart",OBJPROP_PERIOD,InpPeriod);
ObjectSetInteger(0,"Chart",OBJPROP_DATE_SCALE,true);
ObjectSetInteger(0,"Chart",OBJPROP_WIDTH,1);
ObjectSetInteger(0,"Chart",OBJPROP_PRICE_SCALE,true);
ObjectSetInteger(0,"Chart",OBJPROP_SELECTABLE,true);
ObjectSetInteger(0,"Chart",OBJPROP_SELECTED,true);
ObjectSetInteger(0,"Chart",OBJPROP_COLOR,clrBlue);

long chartId=ObjectGetInteger(0,"Chart",OBJPROP_CHART_ID);
ChartApplyTemplate(chartId,"my.tpl");
ChartRedraw(chartId);

//---
   return(INIT_SUCCEEDED);
  }
```

Теперь мы можем перенести на наш графический объект все настройки и индикаторы графика символа.



Присоединив индикатор к графику символа, мы можем нажать на нем правой кнопкой мышки и изменить его свойства, включая его период, размеры и др.

Функция PlaySound

Функция PlaySound воспроизводит звуковой файл.

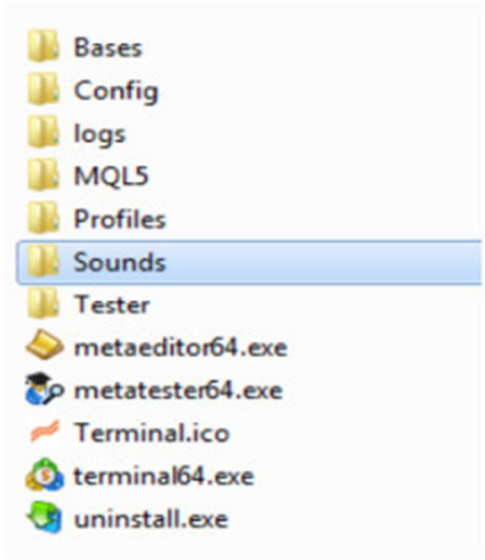
PlaySound

Воспроизводит звуковой файл.

```
bool  PlaySound(
   string  filename      // имя файла
   );
```

Например, это можно делать при появлении сигнала индикатора для напоминания.

В качестве примера добавим звуковой сигнал в наш индикатор Impulse keeper при появлении первого сигнала на покупку или продажу.



Скачаем какой-нибудь WAV-сигнал из Интернета и поместим его файл в папку Sounds терминала.

Добавим код в индикатор Impulse keeper.

```
int countBuy=0;
int countSell=0;

if (start!=1){

if(close[i-1]>open[i-1]&&close[i-1]>EMA34HBuffer[i-1]&&close[i-1]>EMA34LBuffer[i-1]
&&low[i-1]>EMA125Buffer[i-1]&&low[i-1]>PSARBuffer[i-1]&&EMA125Buffer[i-1]<EMA34LBuffer[i-1]
&&EMA125Buffer[i-1]<EMA34HBuffer[i-1]){

     countBuy++;
     if (countBuy==1)PlaySound("alert2.wav");
     }else{
    countBuy=0;
    }

if(close[i-1]<open[i-1]&&close[i-1]<EMA34HBuffer[i-1]&&close[i-1]<EMA34LBuffer[i-1]
&&high[i-1]<EMA125Buffer[i-1]&&high[i-1]<PSARBuffer[i-1]&&EMA125Buffer[i-1]>EMA34LBuffer[i-1]
&&EMA125Buffer[i-1]>EMA34HBuffer[i-1]){

     countSell++;
     if (countSell==1)PlaySound("alert2.wav");
     }else{
    countSell=0;
    }
    }
```

Здесь добавлены счетчики сигналов на продажу и покупку countBuy, countSell, для того, чтобы сигнал звучал только при появлении первого сигнала.

И если появляется сигнал на покупку или продажу, соответствующий счетчик увеличивается и звучит оповещение.

При дублирующем сигнале оповещение не звучит.

При окончании серии сигналов, счетчик обнуляется.

Функция OnChartEvent

Функция OnChartEvent является функцией обратного вызова, которая вызывается при взаимодействии пользователя с графиком символа и событиях, связанных с графическими объектами графика символа.

OnChartEvent

Обработчик события OnChartEvent.

```
virtual void  OnChartEvent(
   const int     id,       // идентификатор события
   const long&   lparam,   // параметр события типа long
   const double  dparam,   // параметр события типа double
   const string  sparam    // параметр события типа string
   )
```

В качестве примера использования функции OnChartEvent рассмотрим наш индикатор Impulse keeper и добавим в него функциональность, позволяющую посмотреть значения используемых индикаторов при клике на сигнале покупки или продажи индикатора.

```
ArraySetAsSeries(time, true);
ArraySetAsSeries(high, true);
ArraySetAsSeries(low, true);
ArraySetAsSeries(open, true);
ArraySetAsSeries(close, true);
ArraySetAsSeries(EMA34HBuffer, true);
ArraySetAsSeries(EMA34LBuffer, true);
ArraySetAsSeries(EMA125Buffer, true);
ArraySetAsSeries(PSARBuffer, true);

for(int i=start;i>=1;i--)
     {
     if(close[i]>open[i]&&close[i]>EMA34HBuffer[i]&&close[i]>EMA34LBuffer[i]
     &&low[i]>EMA125Buffer[i]&&low[i]>PSARBuffer[i]&&EMA125Buffer[i]<EMA34LBuffer[i]
     &&EMA125Buffer[i]<EMA34HBuffer[i]){

     if(!ObjectCreate(0,"Buy"+i,OBJ_ARROW,0,time[i],high[i]))
     {
      return(false);
     }
```

Для этого добавим в код индикатора функцию OnChartEvent, обрабатывающую событие щелчка мыши на графическом объекте индикатора.

Но сначала, здесь мы в функции OnCalculate с помощью функции ArraySetAsSeries изменили порядок доступа ко всем используемым массивам и здесь нам не нужно использовать индекс i-1, так как в текущем тике мы начинаем цикл с индекса 1.

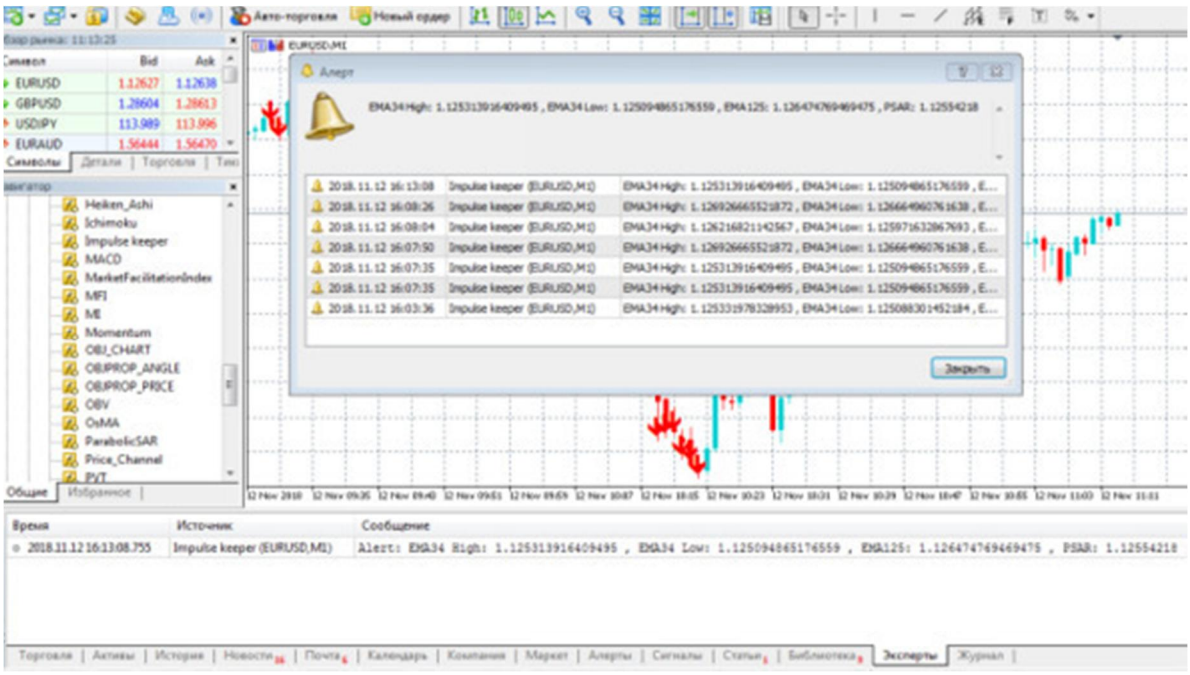
```
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//---
   if(id==CHARTEVENT_OBJECT_CLICK){

  if(StringFind(sparam,"Sell",0)!=-1){
  int pos=StringToInteger(StringSubstr(sparam,4));
Alert("EMA34 High: ", EMA34HBuffer[pos]," , EMA34 Low: ",
EMA34LBuffer[pos], " , EMA125: ", EMA125Buffer[pos], " , PSAR: ", PSARBuffer[pos] );
  }

 if(StringFind(sparam,"Buy",0)!=-1){
  int pos=StringToInteger(StringSubstr(sparam,3));
Alert("EMA34 High: ", EMA34HBuffer[pos]," , EMA34 Low: ",
EMA34LBuffer[pos], " , EMA125: ", EMA125Buffer[pos], " , PSAR: ", PSARBuffer[pos] );
  }
   }
  }
//+------------------------------------------------------------------+
```

Теперь, в функции OnChartEvent код сначала проверяет идентификатор события и если событие – это щелчок мыши на графическом объекте, код переходит к проверке, является ли этот объект графическим объектом индикатора.

Дальше код выделяет из имени объекта его порядковый номер, который соответствует индексу бара, и выводит значения буферов используемых индикаторов в диалоговое окно Alert отображения информации пользователю.

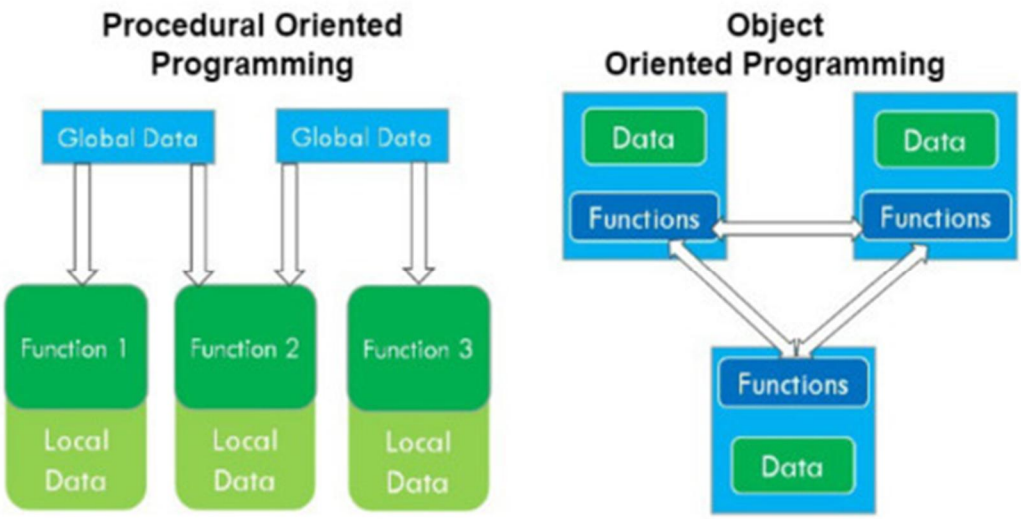


Параллельно информация выводится в окно Эксперты терминала.

Здесь было бы удобно вывести информацию в диалоговое окно MessageBox, которое позволяет взаимодействовать с пользователем, но его нельзя вызывать из пользовательских индикаторов, так как индикаторы выполняются в интерфейсном потоке и не должны его тормозить.

Объектно-ориентированный подход

В объектно-ориентированных языках все – это классы. В том числе и точка входа в приложение является классом, который содержит специфический метод – точку входа в приложение, или который расширяет специфический класс фреймворка или платформы.



С языком MQL5 это немного не так.

Создание программ на языке MQL5 связано с использованием набора функций обратного вызова, которые вызываются клиентским терминалом при наступлении тех или иных событий.

И код MQL5-приложения не является классом, а состоит из набора функций обратного вызова и вспомогательного пользовательского кода.

Так вот в части именно вспомогательного пользовательского кода разработчик и волен использовать либо процедурное программирование, либо объектно-ориентированное программирование.

В случае выбора процедурного программирования вспомогательный пользовательский код представляет собой набор пользовательских функций, при выборе объектно-ориентированного программирования вспомогательный пользовательский код представляет собой набор пользовательских классов, которые могут использовать стандартную MQL5-библиотеку классов.

Напомним базовые понятия объектно-ориентированного программирования.

Инкапсуляция – это когда код представлен классами, которые предоставляют открытые методы для доступа и изменения данных, таким образом защищая данные.

Расширяемость типов – это возможность добавлять пользовательские типы данных, что как раз основано на использовании классов, так как каждый новый пользовательский класс представляет новый тип данных.

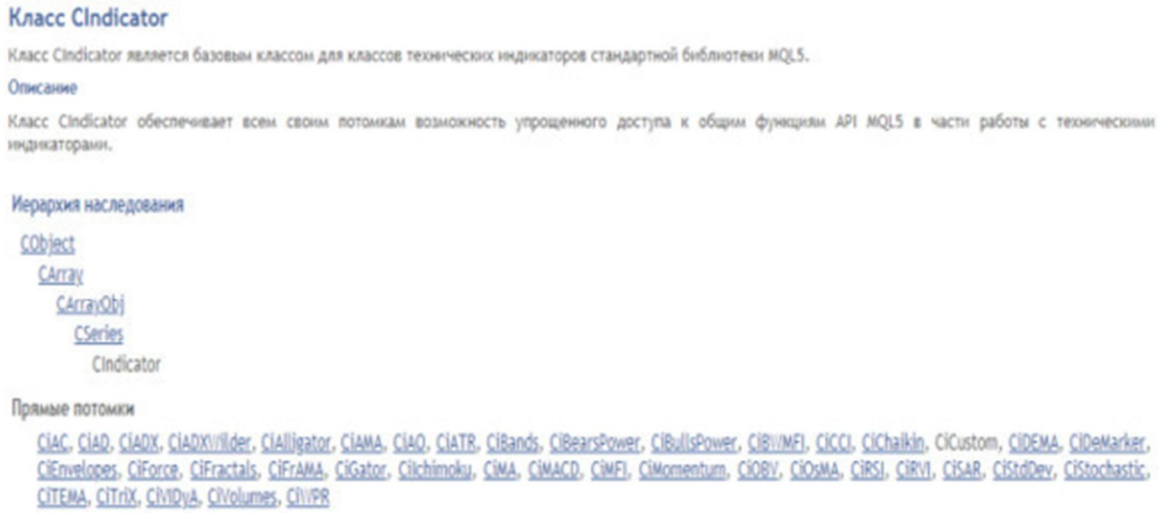
Наследование – это возможность создавать новые классы на основе уже существующих классов, таким образом, повторно используя уже

существующий проверенный и протестированный код. При этом в MQL5 нет множественного наследования.

Полиморфизм – это возможность иметь всем классам одной и той же иерархии наследования метод с одним именем, но разной реализацией.

Перегрузка – это создание методов класса, имеющих одно имя, но предназначенных для работы с разными типами данных, таким образом класс делается универсальным для разных типов данных.

В качестве примера использования объектно-ориентированного подхода рассмотрим создание нашего пользовательского индикатора Impulse keeper с применением классов.


Класс CIndicator

Класс CIndicator является базовым классом для классов технических индикаторов стандартной библиотеки MQL5.

Описание

Класс CIndicator обеспечивает всем своим потомкам возможность упрощенного доступа к общим функциям API MQL5 в части работы с техническими индикаторами.

Иерархия наследования

CObject
CArray
CArrayObj
CSeries
CIndicator

Прямые потомки

CiAC, CiAD, CiADX, CiADXWilder, CiAlligator, CiAMA, CiAO, CiATR, CiBands, CiBearsPower, CiBullsPower, CiBWMFI, CiCCI, CiChaikin, CiCustom, CiDEMA, CiDeMarker, CiEnvelopes, CiForce, CiFractals, CiFrAMA, CiGator, CiIchimoku, CiMA, CiMACD, CiMFI, CiMomentum, CiOBV, CiOsMA, CiRSI, CiRVI, CiSAR, CiStdDev, CiStochastic, CiTEMA, CiTriX, CiVIDyA, CiVolumes, CiWPR

В данном случае, использование класса CIndicator и его наследников CiMA и CiSAR, обеспечивающих доступ к индикаторам MA и PSAR, позволяет вообще обойтись без буферов индикатора Impulse keeper.

Так как они были нужны нам для копирования буферов индикаторов MA и PSAR, а классы CiMA и CiSAR предоставляют напрямую доступ к своим буферам.

```mql5
//+------------------------------------------------------------------+
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#property indicator_chart_window

#include <Indicators\Trend.mqh>

CiMA              MA34H;
CiMA              MA34L;
CiMA              MA125;
CiSAR             SAR;

int    bars_calculated=0;

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
MA34H.Create(_Symbol,PERIOD_CURRENT,34,0,MODE_EMA,PRICE_HIGH);
MA34L.Create(_Symbol,PERIOD_CURRENT,34,0,MODE_EMA,PRICE_LOW);
MA125.Create(_Symbol,PERIOD_CURRENT,125,0,MODE_EMA,PRICE_CLOSE);
SAR.Create(_Symbol,PERIOD_CURRENT,0.02, 0.2);
//---
   return(INIT_SUCCEEDED);
  }
```

Для использования класса CIndicator и его потомков, в код необходимо включить файл Trend.mqh:

Далее в коде индикатора Impulse keeper объявим экземпляры классов CiMA и CiSAR.

И в функции OnInit создадим индикаторы, используя метод Create класса CIndicator.

```mql5
   int start;
   int calculated=MA34H.BarsCalculated();
   if(calculated<=0)
     {
      return(0);
     }
   if(prev_calculated==0 || calculated!=bars_calculated)
     {
      start=rates_total-1;
     }
   else
     {
     start=1;
     }
ArraySetAsSeries(time, true);
ArraySetAsSeries(high, true);
ArraySetAsSeries(low, true);
ArraySetAsSeries(open, true);
ArraySetAsSeries(close, true);

//Print(MA34H.BufferSize());

MA34H.BufferResize(rates_total);
MA34L.BufferResize(rates_total);
MA125.BufferResize(rates_total);
SAR.BufferResize(rates_total);
```

В функции OnCalculate после вычисления начальной позиции расчета индикатора установим размеры буферов индикаторов CiMA и CiSAR, используя метод BufferResize класса CIndicator.

Если это не сделать, размеры буферов используемых индикаторов будут по умолчанию иметь величину 100, и наш индикатор будет рассчитываться только до 100 бара.

Далее обновим данные используемых индикаторов и рассчитаем и отрисуем наш индикатор.

```
for(int i=start;i>=1;i--)
 {
if(close[1]>open[1]&&close[1]>MA34H.Main(1)&&close[1]>MA34L.Main(1)
&&low[1]>MA125.Main(1)&&low[1]>SAR.Main(1)&&MA125.Main(1)<MA34L.Main(1)
&&MA125.Main(1)<MA34H.Main(1)){
 if(!ObjectCreate(0,"Buy"+1,OBJ_ARROW,0,time[1],high[1]))
 {
  return(false);
 }
  ObjectSetInteger(0,"Buy"+1,OBJPROP_COLOR,clrGreen);
  ObjectSetInteger(0,"Buy"+1,OBJPROP_ARROWCODE,233);
  ObjectSetInteger(0,"Buy"+1,OBJPROP_WIDTH,2);
  ObjectSetInteger(0,"Buy"+1,OBJPROP_ANCHOR,ANCHOR_UPPER);
  ObjectSetInteger(0,"Buy"+1,OBJPROP_HIDDEN,true);
  ObjectSetString(0,"Buy"+1,OBJPROP_TOOLTIP,close[1]);
 }
 if(close[1]<open[1]&&close[1]<MA34H.Main(1)&&close[1]<MA34L.Main(1)
 &&high[1]<MA125.Main(1)&&high[1]<SAR.Main(1)&&MA125.Main(1)>MA34L.Main(1)
 &&MA125.Main(1)>MA34H.Main(1)){
 if(!ObjectCreate(0,"Sell"+1,OBJ_ARROW,0,time[1],low[1]))
 {
  return(false);
 }
  ObjectSetInteger(0,"Sell"+1,OBJPROP_COLOR,clrRed);
  ObjectSetInteger(0,"Sell"+1,OBJPROP_ARROWCODE,234);
  ObjectSetInteger(0,"Sell"+1,OBJPROP_WIDTH,2);
  ObjectSetInteger(0,"Sell"+1,OBJPROP_ANCHOR,ANCHOR_LOWER);
  ObjectSetInteger(0,"Sell"+1,OBJPROP_HIDDEN,true);
  ObjectSetString(0,"Sell"+1,OBJPROP_TOOLTIP,close[1]);
 }
 }
 ChartRedraw(0);
```

Чтобы быть последовательными в объектно-ориентированном подходе, весь код по расчету и отрисовке нашего индикатора можно выделить в отдельный пользовательский класс.

Благо мастер редактора MQL5 предоставляет возможность создания каркаса пользовательского класса как опция мастера Новый класс.

```mql5
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#include <Indicators\Trend.mqh>
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
class IKSignal
  {
private:
int _start;
datetime _time[];
double _open[];
double _high[];
double _low[];
double _close[];

public:
                     IKSignal(
                 int start,
                 const datetime &time[],
                 const double &open[],
                 const double &high[],
                 const double &low[],
                 const double &close[]
                  );
bool draw(CiMA &MA34H, CiMA &MA34L, CiMA &MA125, CiSAR &SAR);
                    ~IKSignal();
  };
```

Назовем этот класс как IKSignal.

И здесь, в классе IKSignal мы объявляем закрытые поля класса, представляющие начальную позицию расчета индикатора и ценовую историю.

```mql5
IKSignal::IKSignal(int start,
                 const datetime &time[],
                 const double &open[],
                 const double &high[],
                 const double &low[],
                 const double &close[]
                 )
  {
  _start=start;
  if(ArraySize(time)>0)
     {
     ArrayResize(_time,ArraySize(time));
     ArrayCopy(_time, time);
     }
   if(ArraySize(open)>0)
     {
     ArrayResize(_open,ArraySize(open));
     ArrayCopy(_open, open);
     }
   if(ArraySize(high)>0)
     {
     ArrayResize(_high,ArraySize(high));
     ArrayCopy(_high, high);
     }
   if(ArraySize(low)>0)
     {
     ArrayResize(_low,ArraySize(low));
     ArrayCopy(_low, low);
     }
   if(ArraySize(close)>0)
     {
     ArrayResize(_close,ArraySize(close));
     ArrayCopy(_close, close);
     }
```

```mql5
ArraySetAsSeries(_time, true);
ArraySetAsSeries(_high, true);
ArraySetAsSeries(_low, true);
ArraySetAsSeries(_open, true);
ArraySetAsSeries(_close, true);
  }
```

В конструкторе класса мы копируем его параметры в поля класса и меняем порядок доступа к полям-массивам.

```
bool IKSignal::draw(CiMA &MA34H, CiMA &MA34L, CiMA &MA125, CiSAR &SAR){
for(int i=_start;i>=1;i--)
     {    if(_close[i]>_open[i]&&_close[i]>MA34H.Main(i)&&_close[i]>MA34L.Main(i)
     &&_low[i]>MA125.Main(i)&&_low[i]>SAR.Main(i)&&MA125.Main(i)<MA34L.Main(i)&&MA125.Main(i)<MA34H.Main(i)){
     if(!ObjectCreate(0,"Buy"+i,OBJ_ARROW,0,_time[i],_high[i]))
     {
      return(false);
     }
      ObjectSetInteger(0,"Buy"+i,OBJPROP_COLOR,clrGreen);
      ObjectSetInteger(0,"Buy"+i,OBJPROP_ARROWCODE,233);
      ObjectSetInteger(0,"Buy"+i,OBJPROP_WIDTH,2);
      ObjectSetInteger(0,"Buy"+i,OBJPROP_ANCHOR,ANCHOR_UPPER);
      ObjectSetInteger(0,"Buy"+i,OBJPROP_HIDDEN,true);
      ObjectSetString(0,"Buy"+i,OBJPROP_TOOLTIP,_close[i]);
     }
     if(_close[i]<_open[i]&&_close[i]<MA34H.Main(i)&&_close[i]<MA34L.Main(i)
     &&_high[i]<MA125.Main(i)&&_high[i]<SAR.Main(i)&&MA125.Main(i)>MA34L.Main(i)&&MA125.Main(i)>MA34H.Main(i)){
     if(!ObjectCreate(0,"Sell"+i,OBJ_ARROW,0,_time[i],_low[i]))
     {
      return(false);
     }
      ObjectSetInteger(0,"Sell"+i,OBJPROP_COLOR,clrRed);
      ObjectSetInteger(0,"Sell"+i,OBJPROP_ARROWCODE,234);
      ObjectSetInteger(0,"Sell"+i,OBJPROP_WIDTH,2);
      ObjectSetInteger(0,"Sell"+i,OBJPROP_ANCHOR,ANCHOR_LOWER);
      ObjectSetInteger(0,"Sell"+i,OBJPROP_HIDDEN,true);
      ObjectSetString(0,"Sell"+i,OBJPROP_TOOLTIP,_close[i]);
     }
     }
     ChartRedraw(0);
     return(true);
}
```

Также в классе объявляется открытая функция draw, в которой фактически и будет производиться расчет и отрисовка индикатора.

В качестве параметров этой функции выступают экземпляры классов CiMA и CiSAR.

Тут мы просто переносим код из функции OnCalculate.

Теперь в коде основного файла нам не нужно включать файл Trend.mqh, так как мы уже сделали это в коде класса IKSignal, вместо этого нам нужно включить файл класса IKSignal.

```
#include <IKSignal.mqh>

CiMA              MA34H;
CiMA              MA34L;
CiMA              MA125;
CiSAR             SAR;

int     bars_calculated=0;
```

```

IKSignal iks(start,time,open,high,low,close);

//Print(MA34H.BufferSize());

MA34H.BufferResize(rates_total);
MA34L.BufferResize(rates_total);
MA125.BufferResize(rates_total);
SAR.BufferResize(rates_total);

      MA34H.Refresh();
      MA34L.Refresh();
      MA125.Refresh();
      SAR.Refresh();

      if(!iks.draw(MA34H, MA34L, MA125, SAR)){
      return(false);
      }
```

Поместим файл класса IKSignal в каталог Include и включим его в основной файл индикатора:

Теперь функция OnCalculate примет следующий вид.

Здесь мы создаем экземпляр класса IKSignal с указанными в правильном порядке параметрами и применяем к нему функцию draw.

Как видно, код основного файла индикатора значительно упрощается.



При этом функциональность индикатора осталась той же самой.

Почему не работают индикаторы

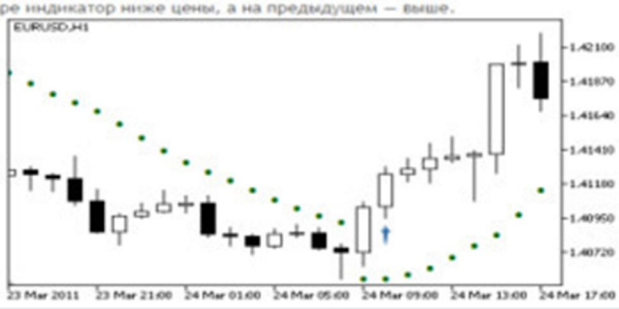
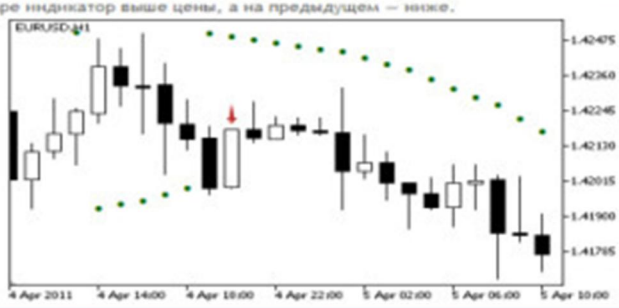
Для торговли на Форекс предлагается огромное число самых разнообразных индикаторов, накладывая которые на график цены сразу же видно, что все они замечательно предсказывают движение цены и дают замечательные сигналы на продажу и покупку.

Почему же тогда начинающий трейдер, пользуясь индикаторами, сразу же уходит в минус?

Основные причины две.

Первая причина – это размер спреда. При увеличении спреда количество удачных сделок будет стремительно уменьшаться.

И вторая причина – это то, что в реале сделка не открывается на открытии того же бара, на котором индикатор дает сигнал, а открывается или закрывается позже.

За покупку	**Разворот** – на анализируемом баре индикатор ниже цены, а на предыдущем – выше. 
За продажу	**Разворот** – на анализируемом баре индикатор выше цены, а на предыдущем – ниже. 

В качестве примера рассмотрим индикатор Parabolic SAR.

Покупка – разворот, на анализируемом баре индикатор ниже цены, а на предыдущем – выше.

Продажа – разворот, на анализируемом баре индикатор выше цены, а на предыдущем – ниже.

Возьмем шаг индикатора 0.02, спред 0.0002 на паре EURUSD.

При условии открытия и закрытия позиции на открытии того же бара, на котором индикатор формирует сигнал, получаются следующие результаты:

Для сделок на покупку:
На периоде M15 получено кол-во Profit(+) 2576, Profit(-) 1659
На периоде M30 получено кол-во Profit(+) 2724, Profit(-) 1455
На периоде H1 получено кол-во Profit(+) 2797, Profit(-) 1449
На периоде H4 получено кол-во Profit(+) 898, Profit(-) 408

Для сделок на продажу:
На периоде M15 получено кол-во Profit(+) 2598, Profit(-) 1637
На периоде M30 получено кол-во Profit(+) 2683, Profit(-) 1496
На периоде H1 получено кол-во Profit(+) 2792, Profit(-)

Здесь для сделок на покупку и продажу на 15 минутных, 30 минутных, часовых и четырех часовых графиках мы получаем прибыльных сделок больше, чем убыточных.

Как видно, на любом интервале индикатор работает, количество сделок с положительным профитом превышает количество сделок с отрицательным профитом, и мы при любом раскладе должны зарабатывать деньги.

Однако в реале сделка открывается и закрывается позже, и результаты станут следующими для открытия и закрытия, например, на следующем баре.



Здесь для сделок на покупку и продажу на 15 минутных, 30 минутных, часовых и четырех часовых графиках мы получаем убыточных сделок больше, чем прибыльных.

Как мы видим, картина резко меняется и на любом интервале индикатор не работает, количество сделок с отрицательным профитом превышает количество сделок с положительным профитом, и мы при любом раскладе теряем деньги.

Конечно такая ситуация зависит от конкретной валютной пары и конкретного периода времени.

На каком-то периоде времени и такой индикатор может показать прибыль.

Но в долгосрочной перспективе, при открытии и закрытии позиции на следующем баре индикатор не покажет итоговый профит.

Поэтому в реальной торговле пользоваться одним индикатором нельзя и нужно разрабатывать сложные торговые стратегии.

Общая структура советника

Советник или эксперт это MQL5-программа, способная автоматически выставлять и закрывать ордера на покупку и продажу финансового инструмента, таким образом, осуществляя автоматическую торговлю в клиентском терминале.

Советник — автоматическая торговая система, имеющая привязку к определенному графику. Советник содержит в себе функции-обработчики предопределенных событий, при наступлении которых выполняются соответствующие элементы торговой стратегии. Примеры таких событий - инициализация и деинициализация программы, приход нового тика, срабатывание таймера, изменение в стакане цен, события графика и пользовательские события.
Советник может не только вычислять торговые сигналы по заложенным правилам, но и автоматически совершать сделки на торговом счете, направляя их прямо на торговый сервер.
Советники хранятся в директории *<каталог_терминала>\MQL5\Experts*.

Подобно индикаторам, код экспертов основан на функциях обратного вызова, вызываемых клиентским терминалом при наступлении определенных событий.

Для эксперта это такие функции как OnInit (), OnDeinit, OnTick (), OnTimer (), OnTrade (), OnTradeTransaction, OnTester (), OnTesterInit (), OnTesterPass (), OnTesterDeinit (), OnBookEvent (), OnChartEvent ().

Впрочем, для организации автоматической торговли достаточно двух функций OnInit () и OnTick ().

OnInit

Вызывается в индикаторах и экспертах при наступлении события Init. Функция предназначена для инициализации запущенной MQL5-программы. Существуют два варианта функции.

Версия с возвратом результата

```
int  OnInit(void);
```

Возвращаемое значение

Значение типа int, ноль означает успешную инициализацию.

Приоритетным является использование вызова OnInit() с возвратом результата выполнения, так как этот способ позволяет не только выполнить инициализацию программы, но и вернуть код ошибки в случае досрочного прекращения программы.

Версия без возврата результата оставлена только для совместимости со старыми кодами. Не рекомендуется к использованию

```
void  OnInit(void);
```

OnTick

Вызывается в экспертах при наступлении события NewTick для обработки новой котировки.

```
void  OnTick(void);
```

Возвращаемое значение

Нет возвращаемого значения

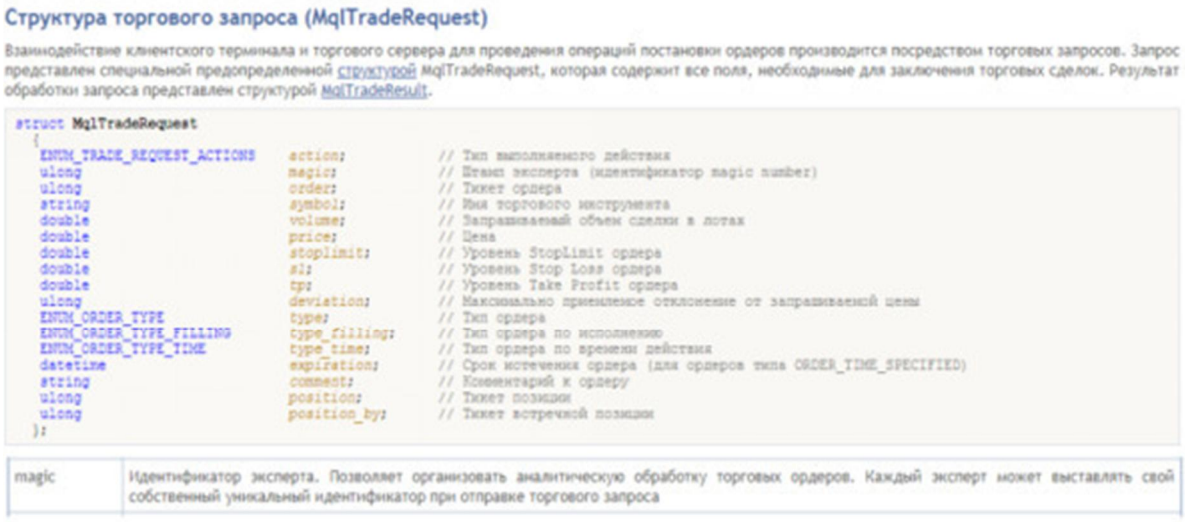
Примечание

Событие NewTick генерируется только для экспертов при поступлении нового тика по символу, к графику которого прикреплен эксперт. Функцию OnTick() бесполезно определять в пользовательском индикаторе или скрипте, поскольку событие NewTick для них не генерируется.

В отличие от индикаторов, для экспертов особо никакие свойства не объявляются, за исключением link, copyright, version и description, и, если эксперт попутно с торговлей не рисует индикатор.

Поэтому перед функциями обратного вызова, в эксперте, объявляются входные параметры, хэндлы используемых технических индикаторов, глобальные переменные и константы.

Здесь правда есть один параметр, которого нет в индикаторе, но который присутствует для эксперта.



Структура торгового запроса (MqlTradeRequest)

Взаимодействие клиентского терминала и торгового сервера для проведения операций постановки ордеров производится посредством торговых запросов. Запрос представлен специальной предопределенной структурой MqlTradeRequest, которая содержит все поля, необходимые для заключения торговых сделок. Результат обработки запроса представлен структурой MqlTradeResult.

```
struct MqlTradeRequest
  {
   ENUM_TRADE_REQUEST_ACTIONS    action;       // Тип выполняемого действия
   ulong                         magic;        // Magic эксперта (идентификатор magic number)
   ulong                         order;        // Тикет ордера
   string                        symbol;       // Имя торгового инструмента
   double                        volume;       // Запрашиваемый объем сделки в лотах
   double                        price;        // Цена
   double                        stoplimit;    // Уровень StopLimit ордера
   double                        sl;           // Уровень Stop Loss ордера
   double                        tp;           // Уровень Take Profit ордера
   ulong                         deviation;    // Максимально приемлемое отклонение от запрашиваемой цены
   ENUM_ORDER_TYPE               type;         // Тип ордера
   ENUM_ORDER_TYPE_FILLING       type_filling; // Тип ордера по исполнению
   ENUM_ORDER_TYPE_TIME          type_time;    // Тип ордера по времени действия
   datetime                      expiration;   // Срок истечения ордера (для ордеров типа ORDER_TIME_SPECIFIED)
   string                        comment;      // Комментарий к ордеру
   ulong                         position;     // Тикет позиции
   ulong                         position_by;  // Тикет встречной позиции
  };
```

magic	Идентификатор эксперта. Позволяет организовать аналитическую обработку торговых ордеров. Каждый эксперт может выставлять свой собственный уникальный идентификатор при отправке торгового запроса

Это магическое число или идентификатор эксперта.

С помощью магического числа идентифицируются торговые ордера, выставляемые экспертом.

Это дает возможность создания взаимосвязанной системы из нескольких работающих экспертов.



```
//+------------------------------------------------------------------+
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---

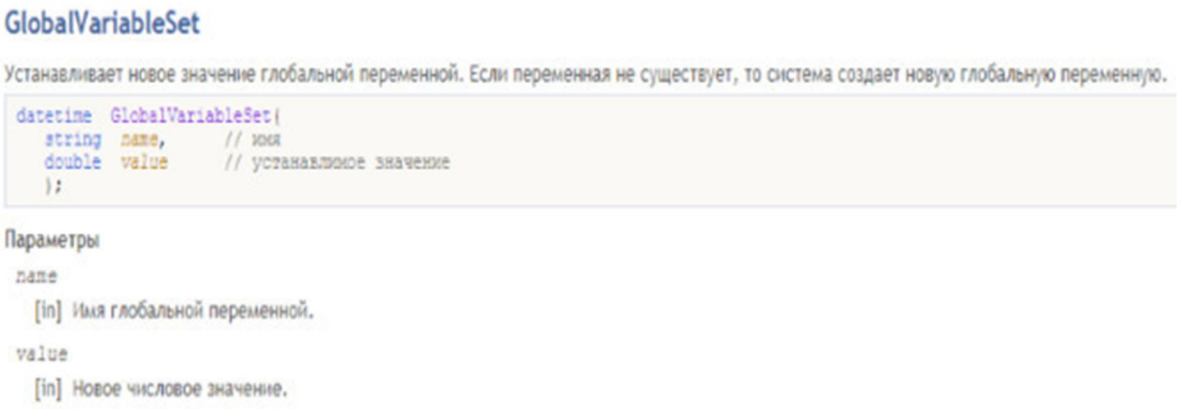
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
```

В функции OnInit эксперта производится инициализация хэндлов используемых технических индикаторов и переменных эксперта.

В функции OnDeinit эксперта, как правило, производится удаление глобальных переменных клиентского терминала, созданных экспертом во время тестирования, оптимизации или отладки, а также освобождение расчетной части используемого индикатора и удаление индикатора из графика по окончании тестирования эксперта с помощью функции IndicatorRelease ().

Глобальные переменные клиентского терминала отличаются от глобальных переменных MQL5-приложения.



GlobalVariableSet

Устанавливает новое значение глобальной переменной. Если переменная не существует, то система создает новую глобальную переменную.

```
datetime  GlobalVariableSet(
   string  name,      // имя
   double  value      // устанавливаемое значение
   );
```

Параметры

name

[in] Имя глобальной переменной.

value

[in] Новое числовое значение.

Глобальные переменные клиентского терминала могут быть созданы MQL5-приложением с помощью функции GlobalVariableSet, но при этом они становятся доступными для других MQL5-приложений клиентского терминала, в отличие от глобальных переменных MQL5-приложения.

Таким образом, глобальные переменные клиентского терминала это средство коммуникации между разными MQL5-приложениями.

Как правило, глобальные переменные клиентского терминала создаются экспертами для проверки на истечение временного лимита для предыдущей сделки.

Сами по себе глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются. Обращением к глобальной переменной считается не только установка нового значения, но и чтение значения глобальной переменной.

При тестировании эксперта глобальные переменные клиентского терминала эмулируются, и они никак не связаны с настоящими глобальными переменными терминала.

Все операции с глобальными переменными терминала при тестировании эксперта производятся вне клиентского терминала в агенте тестирования.

GlobalVariablesDeleteAll

Удаляет глобальные переменные клиентского терминала.

```
int  GlobalVariablesDeleteAll(
   string     prefix_name=NULL,     // все глобальные переменные, чьи имена начинаются с префикса
   datetime   limit_data=0          // все глобальные переменные, которые изменялись ранее ранее указанной даты
   );
```

Параметры

prefix_name=NULL

[in] Префикс имени удаляемых глобальных переменных. Если указан префикс NULL либо пустая строка, то под критерий удаления соответствуют все глобальные переменные, соответствующие критерию удаления по дате

limit_data=0

[in] Дата для отбора глобальных переменных по времени последней модификации. Удаляются глобальные переменные, которые изменялись ранее указанной даты. Если параметр равен нулю, то удаляются все глобальные переменные, соответствующие первому критерию (по префиксу).

Возвращаемое значение

Количество удаленных переменных.

Принудительно глобальные переменные клиентского терминала можно уничтожить с помощью вызова функции GlobalVariableDel или GlobalVariablesDeleteAll.

OnTick

Вызывается в экспертах при наступлении события NewTick для обработки новой котировки.

```
void  OnTick(void);
```

Возвращаемое значение

Нет возвращаемого значения

Примечание

Событие NewTick генерируется только для экспертов при поступлении нового тика по символу, к графику которого прикреплен эксперт. Функцию OnTick() бесполезно определять в пользовательском индикаторе или скрипте, поскольку событие NewTick для них не генерируется.

Событие Tick генерируется только для экспертов, но это не означает, что эксперты обязаны иметь функцию OnTick(), так как для экспертов генерируются не только события NewTick, но и события Timer, BookEvent и ChartEvent.

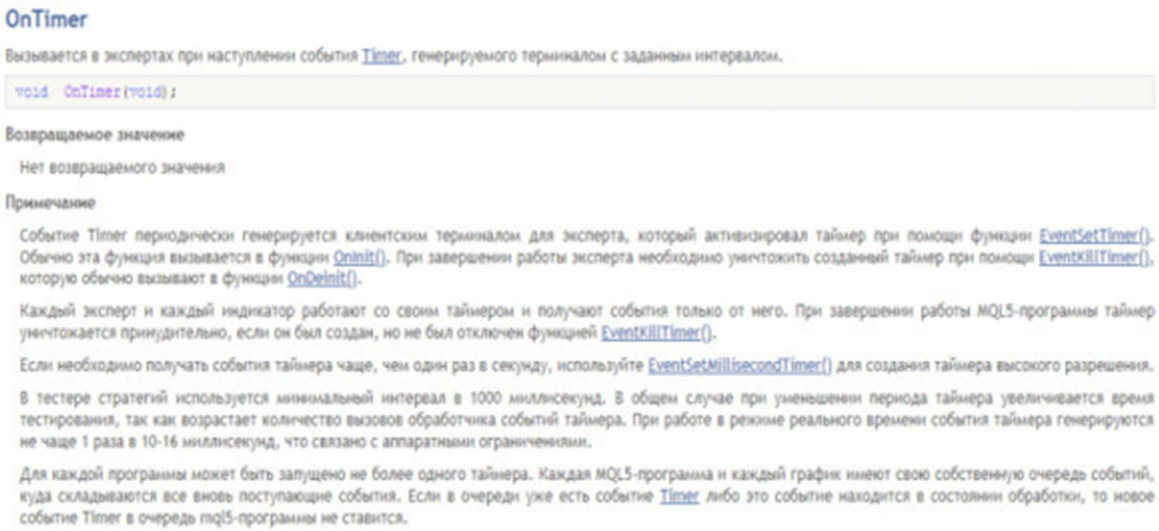
Все события обрабатываются одно за другим в порядке поступления. Если в очереди уже есть событие NewTick либо это событие находится в состоянии обработки, то новое событие NewTick в очередь mql5-программы не ставится.

Событие NewTick генерируется независимо от того, запрещена или разрешена автоматическая торговля (кнопка "Авто-торговля"). Запрет автоматической торговли означает только запрет на отправку торговых запросов из эксперта, работа эксперта при этом не прекращается.

Запрет автоматической торговли путем нажатия кнопки "Авто-торговля" не прерывает текущее выполнение функции OnTick().

В функции OnTick эксперта сначала производится проверка возможности торговли на данном счете, достаточности средств на счете, достаточности загруженной ценовой истории для расчетов торговой стратегии.

Затем устанавливаются временные фильтры для совершения торговых операций, проверяется наличие открытых позиций, производятся вычисления торговой стратегии и на основании ее сигналов открываются или закрываются позиции или выставляются отложенные ордера.



OnTimer

Вызывается в экспертах при наступлении события Timer, генерируемого терминалом с заданным интервалом.

```
void  OnTimer(void);
```

Возвращаемое значение

Нет возвращаемого значения

Примечание

Событие Timer периодически генерируется клиентским терминалом для эксперта, который активизировал таймер при помощи функции EventSetTimer(). Обычно эта функция вызывается в функции OnInit(). При завершении работы эксперта необходимо уничтожить созданный таймер при помощи EventKillTimer(), которую обычно вызывают в функции OnDeinit().

Каждый эксперт и каждый индикатор работают со своим таймером и получают события только от него. При завершении работы MQL5-программы таймер уничтожается принудительно, если он был создан, но не был отключен функцией EventKillTimer().

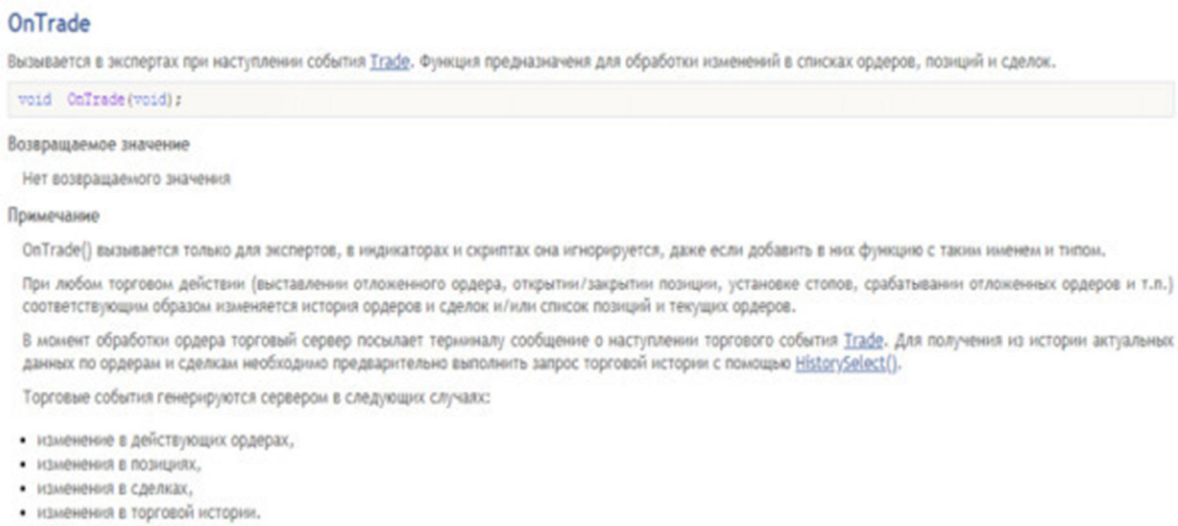
Если необходимо получать события таймера чаще, чем один раз в секунду, используйте EventSetMillisecondTimer() для создания таймера высокого разрешения.

В тестере стратегий используется минимальный интервал в 1000 миллисекунд. В общем случае при уменьшении периода таймера увеличивается время тестирования, так как возрастает количество вызовов обработчика событий таймера. При работе в режиме реального времени события таймера генерируются не чаще 1 раза в 10-16 миллисекунд, что связано с аппаратными ограничениями.

Для каждой программы может быть запущено не более одного таймера. Каждая MQL5-программа и каждый график имеют свою собственную очередь событий, куда складываются все вновь поступающие события. Если в очереди уже есть событие Timer либо это событие находится в состоянии обработки, то новое событие Timer в очередь mql5-программы не ставится.

Функция OnTimer позволяет создать альтернативную модель эксперта, который будет производить вычисления торговой стратегии не при наступлении нового тика в функции OnTick, а через определенные функцией EventSetTimer промежутки времени.

При этом в функции OnDeinit эксперта нужно удалить таймер с помощью вызова функции EventKillTimer.



OnTrade

Вызывается в экспертах при наступлении события Trade. Функция предназначена для обработки изменений в списках ордеров, позиций и сделок.

```
void  OnTrade(void);
```

Возвращаемое значение

Нет возвращаемого значения

Примечание

OnTrade() вызывается только для экспертов, в индикаторах и скриптах она игнорируется, даже если добавить в них функцию с таким именем и типом.

При любом торговом действии (выставлении отложенного ордера, открытии/закрытии позиции, установке стопов, срабатывании отложенных ордеров и т.п.) соответствующим образом изменяется история ордеров и сделок и/или список позиций и текущих ордеров.

В момент обработки ордера торговый сервер посылает терминалу сообщение о наступлении торгового события Trade. Для получения из истории актуальных данных по ордерам и сделкам необходимо предварительно выполнить запрос торговой истории с помощью HistorySelect().

Торговые события генерируются сервером в следующих случаях:

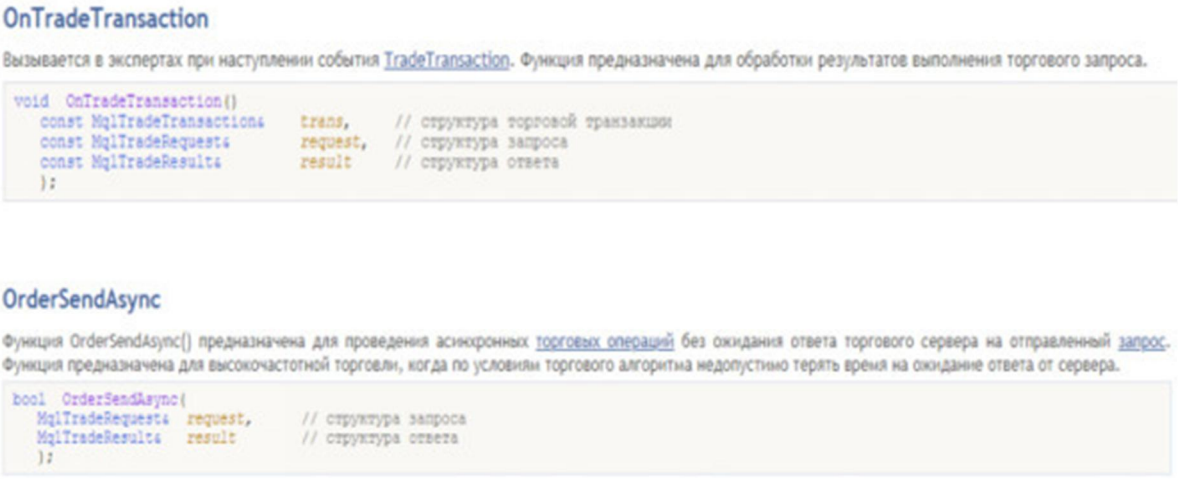
- изменение в действующих ордерах,
- изменения в позициях,
- изменения в сделках,
- изменения в торговой истории.

Функция OnTrade позволяет обработать завершение торговой операции.

Торговая операция это не только открытие или закрытие позиции, это также установка, модификация или удаление отложенного ордера, отмена отложенного ордера при нехватке средств либо по истечении срока действия, срабатывание отложенного ордера, модификация открытой позиции.

Например, функцию OnTrade можно использовать для временного ограничения торговли при срабатывании Stop Loss.

Изменение состояния торгового счета происходит в результате серии транзакций, например, создание ордера, исполнение ордера, удаление ордера из списка открытых, добавление в историю ордеров, добавление сделки в историю, создание новой позиции.



OnTradeTransaction

Вызывается в экспертах при наступлении события TradeTransaction. Функция предназначена для обработки результатов выполнения торгового запроса.

```
void  OnTradeTransaction()
   const MqlTradeTransaction&    trans,     // структура торговой транзакции
   const MqlTradeRequest&        request,   // структура запроса
   const MqlTradeResult&         result     // структура ответа
   );
```

OrderSendAsync

Функция OrderSendAsync() предназначена для проведения асинхронных торговых операций без ожидания ответа торгового сервера на отправленный запрос. Функция предназначена для высокочастотной торговли, когда по условиям торгового алгоритма недопустимо терять время на ожидание ответа от сервера.

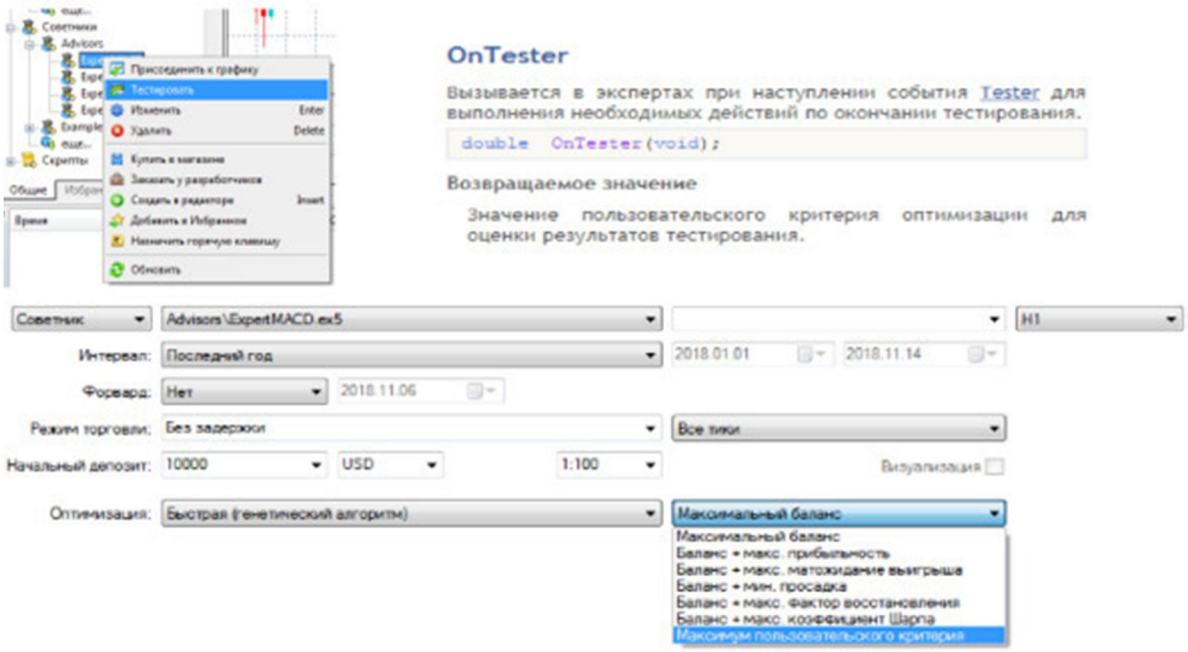
```
bool  OrderSendAsync(
   MqlTradeRequest&  request,     // структура запроса
   MqlTradeResult&   result       // структура ответа
   );
```

Каждая такая транзакция сопровождается вызовом функции OnTradeTransaction, в которой можно обрабатывать выполнение транзакции.

В частности, в функции OnTradeTransaction можно обрабатывать результат исполнения торгового запроса на сервере, отправленного функцией OrderSendAsync.

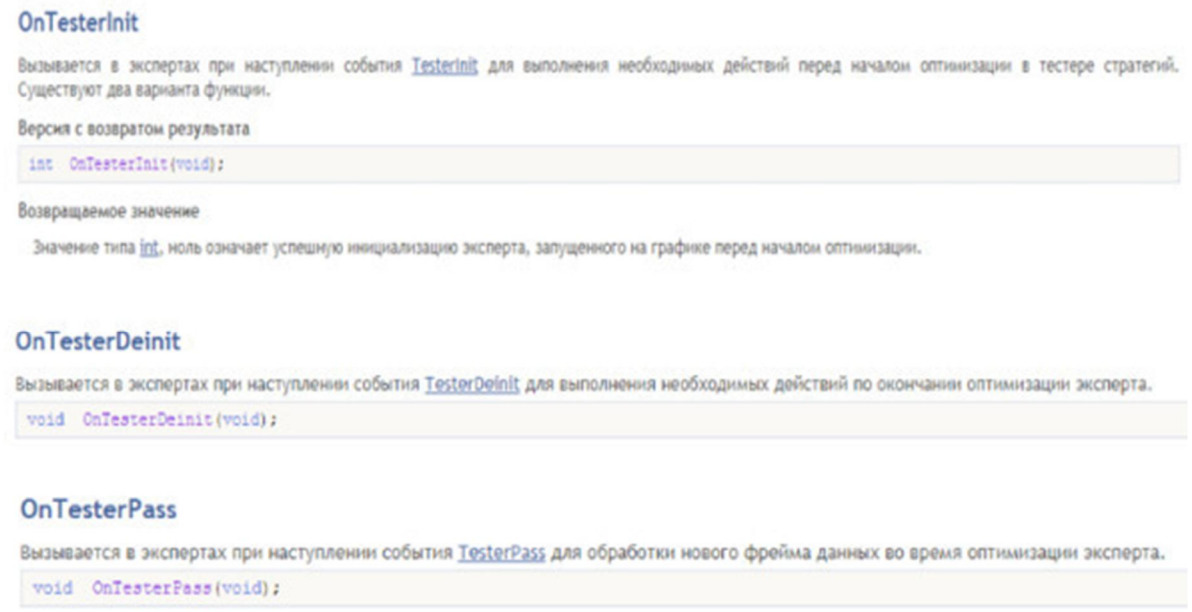
При тестировании эксперта в режиме тестирования с использованием генетического алгоритма производится подбор наилучшей комбинации входных параметров эксперта по критерию оптимизации.

Изначально тестер предлагает набор предопределенных критериев оптимизации, таких как максимальный баланс счета, баланс + максимальная прибыльность, баланс + минимальная просадка и другие.

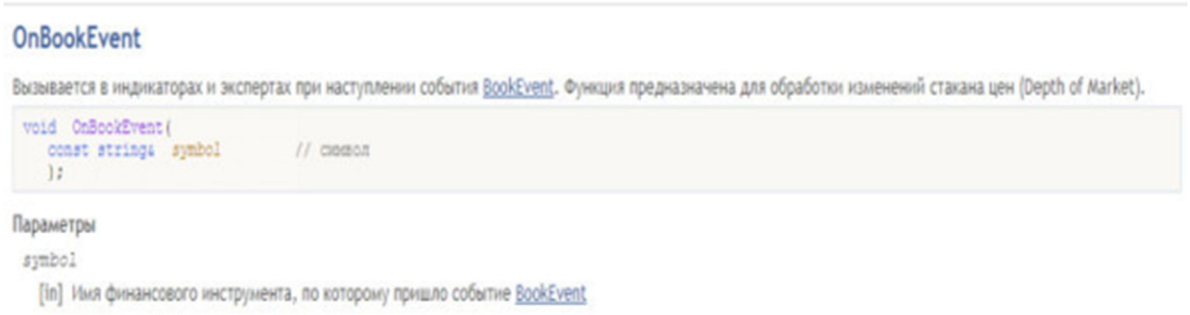


Функция OnTester позволяет определить свой критерий оптимизации, так как при оптимизации тестер всегда ищет локальный максимум возвращаемого значения функции OnTester, которая автоматически вызывается после окончания очередного прохода тестирования эксперта на заданном интервале дат.

Для использования функции OnTester в тестере выбирается критерий оптимизации Максимум пользовательского критерия.



Функции OnTesterInit (), OnTesterPass (), OnTesterDeinit () позволяют организовать динамическую обработку результатов оптимизации параметров эксперта в тестере при каждом проходе оптимизации.



OnBookEvent

Вызывается в индикаторах и экспертах при наступлении события BookEvent. Функция предназначена для обработки изменений стакана цен (Depth of Market).

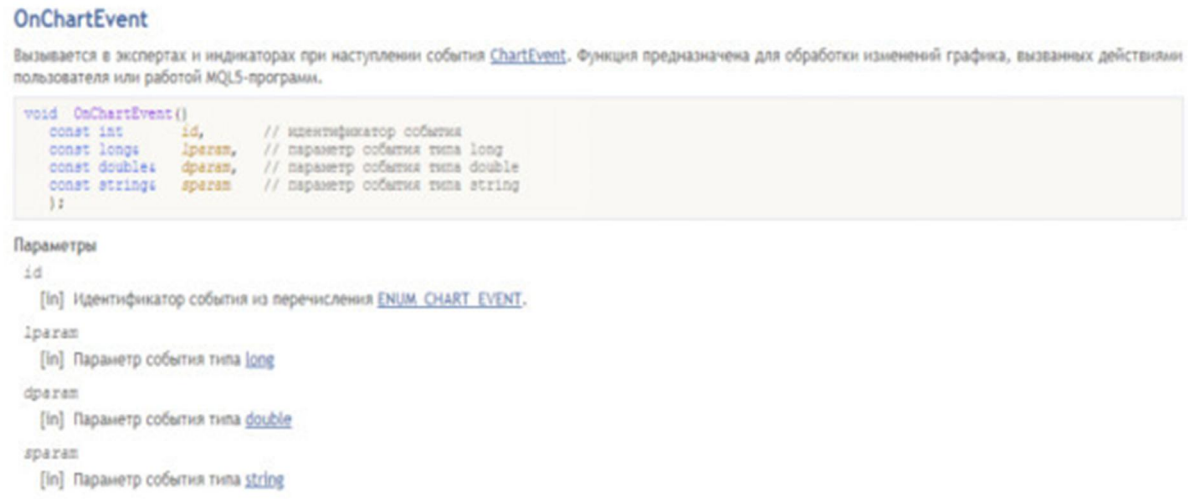
```
void  OnBookEvent(
   const string&  symbol        // символ
   );
```

Параметры

symbol

[in] Имя финансового инструмента, по которому пришло событие BookEvent

Функция OnBookEvent позволяет разработать советник или индикатор, использующий торговую стратегию, которая основана на стакане цен, если конечно дилинговый центр предоставляет такую возможность.



OnChartEvent

Вызывается в экспертах и индикаторах при наступлении события ChartEvent. Функция предназначена для обработки изменений графика, вызванных действиями пользователя или работой MQL5-программ.

```
void  OnChartEvent()
   const int       id,      // идентификатор события
   const long&     lparam,  // параметр события типа long
   const double&   dparam,  // параметр события типа double
   const string&   sparam   // параметр события типа string
   );
```

Параметры

id

[in] Идентификатор события из перечисления ENUM_CHART_EVENT.

lparam

[in] Параметр события типа long

dparam

[in] Параметр события типа double

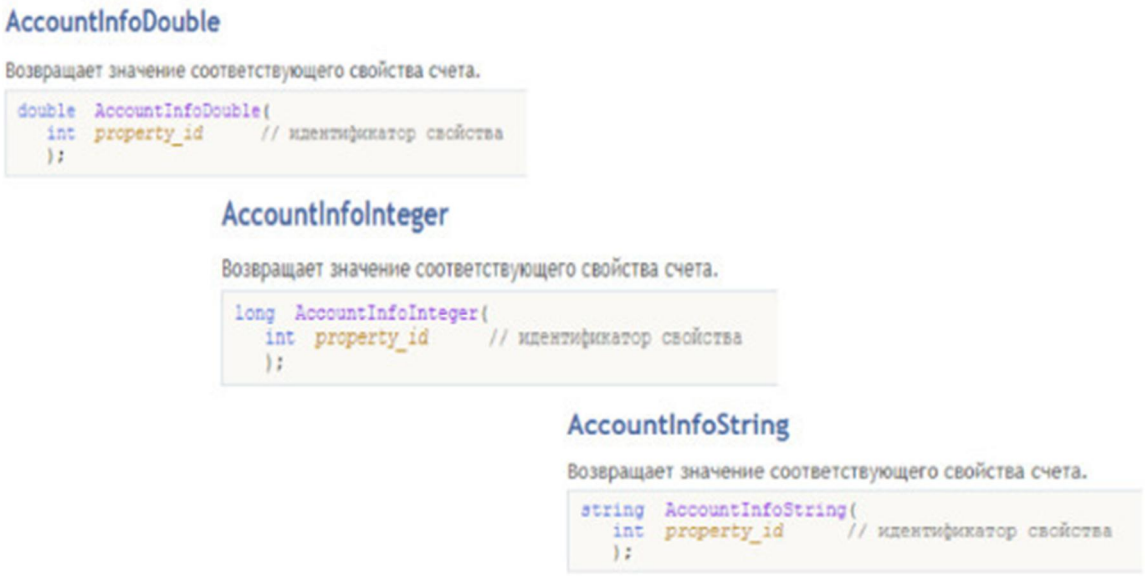
sparam

[in] Параметр события типа string

Функция OnChartEvent, так же как и функция OnTimer, позволяет создать альтернативную модель эксперта, который будет производить вычисления торговой стратегии не при наступлении нового тика в функции OnTick, а при получении событий от индикаторов, прикрепленных к графику символа.

Функция OnTick

Как уже было сказано, в функции OnTick, код, как правило, перед вычислениями торговой стратегии начинается с различного рода проверок, хотя некоторые проверки можно выполнить и в функции OnInit.



Информацию о счете клиента можно получить с помощью функций AccountInfoDouble, AccountInfoInteger и AccountInfoString.

В качестве аргумента этих функций указывается идентификатор свойства, значение которого нужно получить.

Для функции AccountInfoInteger это следующие свойства:

Для функции AccountInfoInteger()

ENUM_ACCOUNT_INFO_INTEGER

Идентификатор	Описание	Тип свойства
ACCOUNT_LOGIN	Номер счета	long
ACCOUNT_TRADE_MODE	Тип торгового счета	ENUM_ACCOUNT_TRADE_MODE
ACCOUNT_LEVERAGE	Размер предоставленного плеча	long
ACCOUNT_LIMIT_ORDERS	Максимально допустимое количество действующих отложенных ордеров	int
ACCOUNT_MARGIN_SO_MODE	Режим задания минимально допустимого уровня залоговых средств	ENUM_ACCOUNT_STOPOUT_MODE
ACCOUNT_TRADE_ALLOWED	Разрешенность торговли для текущего счета	bool
ACCOUNT_TRADE_EXPERT	Разрешенность торговли для эксперта	bool
ACCOUNT_MARGIN_MODE	Режим расчета маржи	ENUM_ACCOUNT_MARGIN_MODE
ACCOUNT_CURRENCY_DIGITS	Количество знаков после запятой для валюты счета, необходимых для точного отображения торговых результатов	int

– ACCOUNT_LOGIN – функция возвращает номер счета.

– ACCOUNT_TRADE_MODE – функция возвращает тип торгового счета. Функция возвращает 0 для демонстрационного торгового счета, 1 для конкурсного торгового счета, 2 для реального торгового счета.

– ACCOUNT_LEVERAGE – возвращает размер кредитного плеча счета, например, для плеча 1:100, функция вернет 100.

– ACCOUNT_LIMIT_ORDERS – функция возвращает максимальное разрешенное количество отложенных ордеров. Такое ограничение устанавливается брокером, и если ограничений нет, функция возвращает 0.

– ACCOUNT_MARGIN_SO_MODE – в чем задается минимально допустимый уровень залоговых средств, в процентах или в деньгах. Минимально допустимый уровень залоговых средств это уровень залоговых средств, при котором требуется или пополнение счета, или уровень залоговых средств, при достижении которого происходит принудительное закрытие самой убыточной позиции. Минимально допустимый уровень залоговых средств устанавливается брокером и функция возвращает 0, если уровень задается в процентах, и возвращает 1, если уровень задается в деньгах.

– ACCOUNT_TRADE_ALLOWED – функция возвращает 0, если для счета запрещена торговля в случае подключения к счету в режиме инвестора, отсутствия соединения к серверу, запрета торговли на стороне сервера, если счет отправлен в архив. Функция возвращает 1, если торговля на счете разрешена.

– ACCOUNT_TRADE_EXPERT – функция возвращает 0, если брокер запрещает автоматическую торговлю, и возвращает 1, если автоматическая торговля разрешена.

Свойство ACCOUNT_LOGIN может быть использовано для защиты эксперта с помощью его привязки к конкретному счету.

```
const long _ACCOUNT=50009917;
. . .
if (AccountInfoInteger(ACCOUNT_LOGIN)!=_ACCOUNT){
  Print("Не соответствует номер счета");
  return(0);
  }
```

Для этого можно объявить константу, представляющую валидный номер счета и в функции OnInit сравнить ее с текущим счетом.

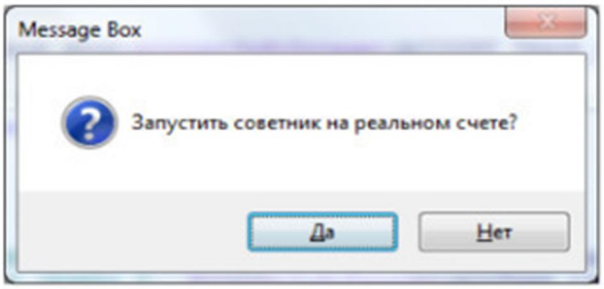
Значение свойства ACCOUNT_TRADE_MODE можно вывести в виде перечисления, для этого возвращаемое функцией значение нужно привести к перечислению, а затем конвертировать в строку.

```
Print("ACCOUNT_TRADE_MODE ", EnumToString
((ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE)));
```

```
if((ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MOD
E_REAL){

int mb=MessageBox("Запустить советник на реальном счете?","Message
Box",MB_YESNO|MB_ICONQUESTION);

    if(mb==IDNO) return(0);
}
```



Свойство ACCOUNT_TRADE_MODE можно использовать для проверки в функции OnInit () запуска эксперта на реальном счете.

Здесь мы сравниваем значение свойства ACCOUNT_TRADE_MODE с ACCOUNT_TRADE_MODE_REAL.

И открываем диалоговое окно для пользователя.

При этом отобразится диалоговое окно, которое при выборе кнопки Да позволит дальнейшее выполнение кода.

```
bool IsNewOrderAllowed(int _max_orders)
  {
int orders=OrdersTotal();
int max_allowed_orders=(int)AccountInfoInteger(ACCOUNT_LIMIT_ORDERS);

if(max_allowed_orders!=0&&_max_orders>max_allowed_orders){
_max_orders=max_allowed_orders;
}

return(orders<_max_orders);
}


 input int max_orders=10; //максимальное количество ордеров

 Print(IsNewOrderAllowed(max_orders));
```

Свойство ACCOUNT_LIMIT_ORDERS может быть использовано для проверки и установки максимального количества отложенных ордеров.

Здесь мы получаем общее количество отложенных ордеров с помощью функции OrdersTotal.

Затем с помощью свойства ACCOUNT_LIMIT_ORDERS получаем максимальное разрешенное количество отложенных ордеров.

И устанавливаем значение максимального количества отложенных ордеров.

Далее мы сравниваем общее количество отложенных ордеров с максимальным количеством отложенных ордеров.

Теперь объявим входной параметр – максимальное количество ордеров, и вызовем определенную нами функцию.

```
//---------------------------------------
if(!TerminalInfoInteger(TERMINAL_CONNECTED)){
Alert("No connection to the trade server");
return(0);
}else{
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED)){
Alert("Trade for this account is prohibited");
return(0);
 }
 }
 if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT)){
 Alert("Trade with the help of experts for the account is prohibited");
 return(0);
 }
//---------------------------------------
```

Проверку свойств ACCOUNT_TRADE_ALLOWED и ACCOUNT_TRADE_EXPERT можно организовать в функции OnInit.

Здесь мы с помощью свойства TERMINAL_CONNECTED проверяем соединение с сервером брокера.

Затем с помощью свойства ACCOUNT_TRADE_ALLOWED проверяем возможность торговли для данного счета.

И с помощью свойства ACCOUNT_TRADE_EXPERT проверяем возможность автоматической торговли.

Дополнительно отдельно проверку соединения с сервером можно сделать в функции OnTick.

Для функции AccountInfoDouble()

ENUM_ACCOUNT_INFO_DOUBLE

Print(AccountInfoDouble(ACCOUNT_BALANCE));

Идентификатор	Описание	Тип свойства
ACCOUNT_BALANCE	Баланс счета в валюте депозита	double
ACCOUNT_CREDIT	Размер предоставленного кредита в валюте депозита	double
ACCOUNT_PROFIT	Размер текущей прибыли на счете в валюте депозита	double
ACCOUNT_EQUITY	Значение собственных средств на счете в валюте депозита	double
ACCOUNT_MARGIN	Размер зарезервированных залоговых средств на счете в валюте депозита	double
ACCOUNT_MARGIN_FREE	Размер свободных средств на счете в валюте депозита, доступных для открытия позиции	double
ACCOUNT_MARGIN_LEVEL	Уровень залоговых средств на счете в процентах	double
ACCOUNT_MARGIN_SO_CALL	Уровень залоговых средств, при котором требуется пополнение счета (Margin Call). В зависимости от установленного ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита	double
ACCOUNT_MARGIN_SO_SO	Уровень залоговых средств, при достижении которого происходит принудительное закрытие самой убыточной позиции (Stop Out). В зависимости от установленного ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита	double
ACCOUNT_MARGIN_INITIAL	Размер средств, зарезервированных на счёте, для обеспечения гарантийной суммы по всем отложенным ордерам	double
ACCOUNT_MARGIN_MAINTENANCE	Размер средств, зарезервированных на счёте, для обеспечения минимальной суммы по всем открытым позициям	double
ACCOUNT_ASSETS	Текущий размер активов на счёте	double
ACCOUNT_LIABILITIES	Текущий размер обязательств на счёте	double
ACCOUNT_COMMISSION_BLOCKED	Текущая сумма заблокированных комиссий по счёту	double

Для функции AccountInfoDouble определены следующие свойства.

ACCOUNT_BALANCE – баланс счета. Соответствует значению Баланс вкладке Торговля клиентского терминала.



ACCOUNT_CREDIT – размер предоставленного кредита. Типичная ситуация, когда это значение равно 0.

ACCOUNT_PROFIT – размер текущей прибыли на счете. Соответствует столбцу Прибыль во вкладке Торговля клиентского терминала.

ACCOUNT_EQUITY – значение собственных средств на счете. Соответствует значению Средства вкладке Торговля клиентского терминала.

ACCOUNT_MARGIN – размер зарезервированных залоговых средств на счете. Соответствует значению Маржа вкладке Торговля клиентского терминала. Если открытых позиций нет, это значение равно 0.

ACCOUNT_MARGIN_FREE – размер свободных средств на счете, доступных для открытия позиции. Соответствует значению Свободная маржа вкладке Торговля клиентского терминала.

ACCOUNT_MARGIN_LEVEL – уровень залоговых средств на счете в процентах. Соответствует значению Уровень маржи вкладке Торговля клиентского терминала. Рассчитывается как Средства/Маржа*100%. Если открытых позиций нет, это значение равно 0.

```
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT)

 Print(AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL)," %");


if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY)

 Print(AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL)," USD");
```

ACCOUNT_MARGIN_SO_CALL – уровень залоговых средств, при котором требуется пополнение счета (Margin Call).

В зависимости от установленного ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита.

Margin Call это скорее информационный сигнал для трейдера, что его счет близок к закрытию, и не сопровождается действиями брокера.

Последствия наступают в случае возникновения Stop Out.

Например, при ACCOUNT_MARGIN_SO_CALL = 50%, событие Margin Call наступит, когда размер средств на счете станет как половина от маржи.

```
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT)

 Print(AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)," %");


if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY)

 Print(AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)," USD");
```

ACCOUNT_MARGIN_SO_SO – уровень залоговых средств, при достижении которого происходит принудительное закрытие самой убыточной позиции (Stop Out).

В зависимости от установленного ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита.

Например, при ACCOUNT_MARGIN_SO_SO = 10%, событие Stop Out наступит, когда размер средств на счете будет 10% от маржи, при этом открытые позиции начнут принудительно закрываться брокером.

Другие свойства функции AccountInfoDouble.

ACCOUNT_MARGIN_INITIAL – размер средств, зарезервированных на счёте, для обеспечения гарантийной суммы по всем отложенным ордерам.

Как правило, эта величина равна 0.

ACCOUNT_MARGIN_MAINTENANCE – размер средств, зарезервированных на счёте, для обеспечения минимальной суммы по всем открытым позициям.

Как правило, эта величина равна 0.

ACCOUNT_ASSETS – текущий размер активов на счёте.

Как правило, эта величина равна 0.

ACCOUNT_LIABILITIES – текущий размер обязательств на счёте.

Как правило, эта величина равна 0.

ACCOUNT_COMMISSION_BLOCKED – текущая сумма заблокированных комиссий по счёту.

Как правило, эта величина равна 0.

С помощью свойств функции AccountInfoDouble можно организовать различного рода проверки в функции OnTick эксперта.

Margin Call

```
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT){

if(AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)!=0
&&AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL))
Alert("Margin Call!!!");
}

if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY){

if(AccountInfoDouble(ACCOUNT_EQUITY)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL))
Alert("Margin Call!!!");
}
```

Например, наступление события Margin Call.

Здесь мы сравниваем значение свойства ACCOUNT_MARGIN_SO_CALL с размером средств на счете.

Stop Out

```
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT){

if(AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)!=0
&&AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)){
Alert("Stop Out!!!");
return;
}}

if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN
_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY){

if(AccountInfoDouble(ACCOUNT_EQUITY)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)){
Alert("Stop Out!!!");
return;
} }
```

При проверке наступления события Stop Out, мы сравниваем значение свойства ACCOUNT_MARGIN_SO_SO с размером средств на счете.

```
double margin;
MqlTick last_tick;
ResetLastError();

if(SymbolInfoTick(Symbol(),last_tick))
   {
if(OrderCalcMargin(ORDER_TYPE_BUY, Symbol(), Lot, last_tick.ask, margin))
     {
    if(margin>AccountInfoDouble(ACCOUNT_MARGIN_FREE)){
     Alert("Not enough money in the account!");
     return;
    }
      }
    }
  else
    {
     Print(GetLastError());
    }
```

Также можно организовать проверку размера свободных средств на счете, доступных для открытия позиции.

Здесь MqlTick это стандартная структура для хранения цен, которая заполняется значениями с помощью функции SymbolInfoTick.

Вызов функции ResetLastError () производится для обнуления ошибки перед вызовом функции, после которой проверяется возникновение ошибки.

Функция OrderCalcMargin вычисляет размер средств, необходимых для открытия позиции.

И если размер свободных средств на счете (ACCOUNT_MARGIN_FREE) меньше, чем размер средств, необходимых для открытия позиции, денег на счете недостаточно и торговля невозможна.

Для функции AccountInfoString()

ENUM_ACCOUNT_INFO_STRING

Идентификатор	Описание	Тип свойства
ACCOUNT_NAME	Имя клиента	string
ACCOUNT_SERVER	Имя торгового сервера	string
ACCOUNT_CURRENCY	Валюта депозита	string
ACCOUNT_COMPANY	Имя компании, обслуживающей счет	string

Для функции AccountInfoString определены такие свойства как имя клиента ACCOUNT_NAME, имя торгового сервера ACCOUNT_SERVER, валюта депозита ACCOUNT_CURRENCY, имя компании, обслуживающей счет ACCOUNT_COMPANY.

```
if (AccountInfoString(ACCOUNT_NAME)!=_name){
  Print("Не соответствует имя пользователя");
  return(0);
  }
```

С помощью свойства ACCOUNT_NAME, также как и с помощью свойства ACCOUNT_LOGIN, можно защитить советник.

TerminalInfoInteger

Возвращает значение соответствующего свойства окружения mql5-программы.

```
int  TerminalInfoInteger(
   int  property_id      // идентификатор свойства
   );
```

TerminalInfoString

Функция возвращает значение соответствующего свойства окружения mql5-программы. Свойство должно быть типа string

```
string  TerminalInfoString(
   int  property_id      // идентификатор свойства
   );
```

```
#import "dll_lib.dll"

TERMINAL_DLLS_ALLOWED

Print((bool)TerminalInfoInteger(TERMINAL_DLLS_ALLOWED ));
```

Информацию о клиентском терминале можно получить с помощью функций TerminalInfoInteger и TerminalInfoString.

В качестве аргумента эти функции также принимают свойства.

Мы уже видели проверку подключения терминала к серверу с помощью свойства TERMINAL_CONNECTED.

Свойство TERMINAL_DLLS_ALLOWED позволяет выяснить, есть ли разрешение на использование DLL.

Файлы DLL это еще один способ создания повторно используемых библиотек – модулей кода для MQL5-программ.

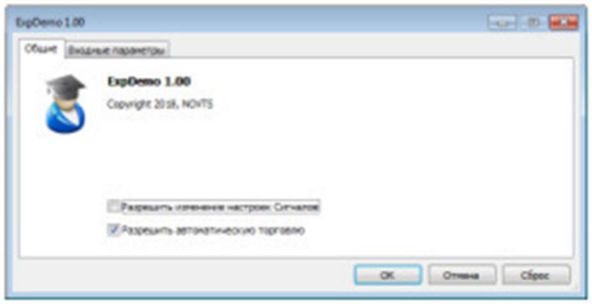
DLL-библиотеки находятся в папке MQL5\Libraries торгового терминала и включаются в код MQL5-программы с помощью команды #import.

При этом разрешение на использование DLL-библиотек устанавливается во вкладке Советники настроек клиентского терминала.

DLL-библиотеки могут также применяться для защиты эксперта с помощью переноса основного кода торговой стратегии в DLL-файл.




```
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
  Alert("Разрешение на автоматическую торговлю выключено!");
```




```
if(!MQLInfoInteger(MQL_TRADE_ALLOWED))
Alert("Автоматическая торговля запрещена в свойствах эксперта ",__FILE__);
```

Свойство TERMINAL_TRADE_ALLOWED показывает, включена ли кнопка авто-торговли в клиентском терминале.

Для проверки этого свойства в функцию OnTick () можно включить код, использующий свойство TERMINAL_TRADE_ALLOWED.

Однако разрешение на торговлю с помощью эксперта может быть отключено в общих свойствах самого эксперта.

Для проверки этого условия можно использовать свойство MQL_TRADE_ALLOWED функции MQLInfoInteger.

SymbolInfoInteger

Возвращает соответствующее свойство указанного символа. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
long  SymbolInfoInteger(
   string                 name,     // символ
   ENUM_SYMBOL_INFO_INTEGER prop_id   // идентификатор свойства
   );
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в принимающую переменную, передаваемую по ссылке последним параметром.

```
bool  SymbolInfoInteger(
   string                 name,     // символ
   ENUM_SYMBOL_INFO_INTEGER prop_id,  // идентификатор свойства
   long&                  long_var  // сюда примем значение свойства
   );
```

```
double _spread = SymbolInfoInteger(Symbol(),SYMBOL_SPREAD)
*MathPow(10, -SymbolInfoInteger(Symbol(),SYMBOL_DIGITS))/MathPow(10,-4);

if(_spread>spreadLevel){
Alert("Слишком большой спред!");
return;
}
```

С помощью свойства SYMBOL_SPREAD функции SymbolInfoInteger можно осуществить контроль над спредом брокера:

Здесь с помощью свойства SYMBOL_DIGITS выясняем, сколько знаков после запятой в цене и вычисляем спред в пунктах.

Затем сравниваем его с пороговым значением, и, если текущий спред больше порогового значения, торговлю не осуществляем.

```
if((ENUM_SYMBOL_TRADE_MODE)SymbolInfoInteger(Symbol(),SYMBOL_TRA
DE_MODE)!=SYMBOL_TRADE_MODE_FULL){

Alert("Установлены ограничения на торговые операции");
return;
}
```

С помощью свойства SYMBOL_TRADE_MODE функции SymbolInfoInteger можно проверить ограничения на торговые операции по символу, установленные брокером.

Здесь мы сравниваем значение свойства SYMBOL_TRADE_MODE с константой SYMBOL_TRADE_MODE_FULL.

Например, по какому-либо финансовому инструменту брокер может отключить торговлю.

```
input int startHour=0; //start hour
input int startMin=0; //start minute
input int stopHour=23; // stop hour
input int stopMin=0; // stop minute
. . .
long _seconds=TimeLocal()%86400;

long startSec=3600*startHour+60*startMin;

long stopSec=3600*stopHour+60*stopMin;

if(_seconds<startSec || _seconds>stopSec)return;
```

Также в функции OnTick можно ограничить работу эксперта по времени.

Если вы хотите, чтобы эксперт работал каждый день в заданный интервал времени, определите начальные и конечные час и минуты временного интервала и сравните их с текущим временем.

Здесь берется локальное время, если вы хотите сравнивать с серверным временем, используйте функцию TimeCurrent, а не функцию TimeLocal.

```
input datetime startSession=D'2015.05.05';
input datetime stopSession=D'2015.05.06';
. . .
datetime  _session = TimeLocal();
if(_session<startSession || _session>stopSession)return;
```

Если вы хотите, чтобы эксперт просто отработал в заданный интервал времени, определите начальную и конечную даты временного интервала и сравните их с текущим временем.

В функции OnTick эксперта также было бы неплохо проверить, достаточно ли баров в истории для расчета советника.

```
if(Bars(Symbol(), 0)<100)
    {
     Alert("In the chart little bars, Expert will not work!!");
     return;
    }

Или


if(SeriesInfoInteger(Symbol(),0,SERIES_BARS_COUNT)<100)
    {
     Alert("In the chart little bars, Expert will not work!!");
     return;
    }
```

Сделать это можно двумя способами – с помощью функции Bars и с помощью свойства SERIES_BARS_COUNT функции SeriesInfoInteger.

Ограничить вычисления советника в функции OnTick по появлению нового бара на графике также можно двумя способами, с помощью свойства SERIES_LASTBAR_DATE функции SeriesInfoInteger или с помощью функции CopyTime.

```
static datetime last_time;
datetime last_bar_time=
(datetime)SeriesInfoInteger(Symbol(),Period(),
SERIES_LASTBAR_DATE);

if(last_time!=last_bar_time)
{
last_time=last_bar_time;

}else{
return;
}
```

```
static datetime Old_Time;
datetime New_Time[1];

int copied=CopyTime(Symbol(),Period(),0,1,New_Time);
 ResetLastError();
  if(copied>0)
   {
    if(Old_Time!=New_Time[0])
     {
     Old_Time=New_Time[0];
     }else{
     return;
     }
   }
  else
   {
    Print(GetLastError());
    return;
   }
```

То есть здесь мы проводим вычисления в функции OnTick только при появлении нового бара на графике символа, пропуская все промежуточные тики.

Делаем мы это, получая время открытия последнего бара и используя статическую локальную переменную для сравнения.

Если вы, предположим, хотите, после того как поймали StopLoss, прекратить на сегодня торговлю советником, вам нужно правильно обработать это StopLoss событие.

```
void OnTrade()
 {
static int _deals;
ulong _ticket=0;

if(HistorySelect(0,TimeCurrent()))
 {
 int  i=HistoryDealsTotal()-1;

  if(_deals!=i) {
  _deals=i;
  } else { return; }

  if((_ticket=HistoryDealGetTicket(i))>0)
   {
    string
_comment=HistoryDealGetString(_ticket,DEAL_COMMENT);

    if(StringFind(_comment,"sl",0)!=-1) {
    flagStopLoss=true;
    }

   } } }
```

Как известно, функция OnTrade вызывается при открытии или закрытии позиции, установке, модификации или удалении отложенного ордера, отмене отложенного ордера при нехватке средств либо по истечении срока действия, срабатывании отложенного ордера, модификации открытой позиции.

Поэтому в функции OnTrade нужно выделить только события совершения сделок, а затем из сделок выделить событие закрытия позиции по StopLoss:

Здесь функция HistorySelect запрашивает историю сделок и ордеров за все время, затем с помощью функции HistoryDealsTotal мы получаем индекс последней сделки и сравниваем его со статической переменной, хранящей индекс предыдущей сделки. Таким образом, мы выделяем только события совершения сделок.

С помощью функции HistoryDealGetTicket получаем тикет последней сделки и, используя свойство DEAL_COMMENT функции HistoryDealGetString, получаем комментарий к сделке.

Если комментарий содержит sl, тогда это была сделка закрытия позиции по StopLoss.

flagStopLoss это глобальная переменная, которую мы теперь можем использовать в функции OnTick.

```
bool flagStopLoss=false;
...
static datetime Old_TimeD1;
datetime New_TimeD1[1];
bool IsNewBarD1=false;

int copiedD1=CopyTime(_Symbol,PERIOD_D1,0,1,New_TimeD1);
ResetLastError();
if(copiedD1>0)
   {
    if(Old_TimeD1!=New_TimeD1[0])
     {
      IsNewBarD1=true;
      Old_TimeD1=New_TimeD1[0];
      flagStopLoss=false;
     }
   }
  else
   {
    Print(GetLastError());
    return;
   }
```

```
if(IsNewBarD1==false)
   {
   if(flagStopLoss==true){
   return;
   }
   }
```

Здесь эксперт прекращает вычисления, если получен StopLoss и не наступил новый дневной бар.

Это флаги flagStopLoss и IsNewBarD1 соответственно.

```
bool BuyOpened=false;
bool SellOpened=false;

  if(PositionSelect(_Symbol)==true)
   {

if(PositionGetInteger(POSITION_TYPE)==POSITION_TY
PE_BUY)
      {
       BuyOpened=true;
      }
    else
if(PositionGetInteger(POSITION_TYPE)==POSITION_TY
PE_SELL)
      {
       SellOpened=true;
      }
   }
```

Так как для каждого финансового инструмента (символа) возможна только одна открытая позиция, в функции OnTick нужно организовать проверку наличия открытой позиции, чтобы не пытаться открыть ее заново, при ее фиксированном объеме.

Здесь функция PositionSelect копирует данные об открытой позиции символа в программное окружение. Затем, с помощью свойства POSITION_TYPE функции PositionGetInteger, выясняется, является ли открытая позиция позицией на продажу или покупку.

Позиции – это наличие купленных или проданных контрактов по финансовому инструменту.

Длинная позиция (Long) образуется в результате покупок в ожидании повышения цены, короткая позиция (Short) – результат продажи актива в расчете на снижение цены в будущем.

На одном счете по каждому финансовому инструменту может существовать только одна позиция.

По каждому символу в любой момент времени может быть только одна открытая позиция – длинная или короткая.

Объем позиции может увеличиваться в результате новой торговой операции в том же направлении.

То есть объем длинной позиции будет увеличен после новой покупки (операции Buy) и уменьшится после продажи (операции Sell).

Позиция считается закрытой, если в результате торговой операции объем обязательств стал равен нулю.

Такая операция называется закрытием позиции.

После выполнения различных проверок в функции OnTick следуют вычисления сигналов торговой системы эксперта.

```
MqlRates mrate[];
ResetLastError();

if(CopyRates(Symbol(), Period(), 0, 3, mrate) < 0)
    {
Print(GetLastError());
    return;
    }
ArraySetAsSeries(mrate,true);
```

Как правило, для вычисления сигналов торговой системы требуются исторические данные символа.

Сделать это можно с помощью функции CopyRates и структуры MqlRates, содержащей исторические цены бара символа.

Здесь в массив структуры MqlRates копируются данные последних трех баров, а затем меняется порядок доступа к массиву.

Наконец, после вычислений сигналов торговой системы эксперта можно открывать или закрывать позиции.

```
if(Lot<SymbolInfoDouble(Symbol(),SYMBOL_VOLUME_MIN)
||Lot>SymbolInfoDouble(Symbol(),SYMBOL_VOLUME_MAX))
return;
```

Но перед совершением сделки было бы неплохо проверить корректность объема, с которым мы собираемся выйти на рынок.

Сделать это можно, используя свойства SYMBOL_VOLUME_MIN и SYMBOL_VOLUME_MAX.

OrderSend

Функция OrderSend() предназначена для совершения торговых операций через отправку запросов на торговый сервер.

```
bool  OrderSend(
   MqlTradeRequest&  request,     // структура запроса
   MqlTradeResult&   result       // структура ответа
   );
```

Параметры

request

[in] Указатель на структуру типа MqlTradeRequest, описывающую торговое действие клиента.

result

[in,out] Указатель на структуру типа MqlTradeResult, описывающую результат торговой операции в случае успешного выполнения (возврата true).

Возвращаемое значение

В случае успешной базовой проверки структур (проверка указателей) возвращается true - **это не свидетельствует об успешном выполнении торговой операции.** Для получения более подробного описания результата выполнения функции следует анализировать поля структуры *result*.

Открытие и закрытие позиции, изменение объема открытой позиции, изменение значения Stop Loss и Take Profit у открытой позиции, установка, модификация и удаление отложенных ордеров, все это может быть сделано с помощью функции OrderSend.

Тип торговой операции, которую будет пытаться выполнить функция OrderSend, определяется структурой MqlTradeRequest.

```
struct MqlTradeRequest
  {
   ENUM_TRADE_REQUEST_ACTIONS    action;       // Тип выполняемого действия
   ulong                         magic;        // Штамп эксперта (идентификатор magic number)
   ulong                         order;        // Тикет ордера
   string                        symbol;       // Имя торгового инструмента
   double                        volume;       // Запрашиваемый объем сделки в лотах
   double                        price;        // Цена
   double                        stoplimit;    // Уровень StopLimit ордера
   double                        sl;           // Уровень Stop Loss ордера
   double                        tp;           // Уровень Take Profit ордера
   ulong                         deviation;    // Максимально приемлемое отклонение от запрашиваемой цены
   ENUM_ORDER_TYPE               type;         // Тип ордера
   ENUM_ORDER_TYPE_FILLING       type_filling; // Тип ордера по исполнению
   ENUM_ORDER_TYPE_TIME          type_time;    // Тип ордера по времени действия
   datetime                      expiration;   // Срок истечения ордера (для ордеров типа ORDER_TIME_SPECIFIED)
   string                        comment;      // Комментарий к ордеру
   ulong                         position;     // Тикет позиции
   ulong                         position_by;  // Тикет встречной позиции
  };
```

Первый параметр action структуры MqlTradeRequest определяет тип торговой операции функции OrderSend с помощью перечисления ENUM_TRADE_REQUEST_ACTIONS.

Это может быть немедленное совершение сделки на покупку или продажу (TRADE_ACTION_DEAL), изменение значений Stop Loss и Take Profit у открытой позиции (TRADE_ACTION_SLTP), установка отложенного ордера на покупку или продажу (TRADE_ACTION_PENDING), изменение параметров отложенного ордера (TRADE_ACTION_MODIFY), удаление отложенного ордера (TRADE_ACTION_REMOVE).

Если вы хотите совершить немедленную сделку на покупку или продажу, в этом случае тип исполнения ордера для данного финансового инструмента или символа определяется брокером.

Это может быть немедленное исполнение (Instant Execution), исполнение по запросу (Request Execution), исполнение по рынку (Market Execution), биржевое исполнение (Exchange Execution).

Выяснить тип исполнения ордера можно с помощью свойства SYMBOL_TRADE_EXEMODE функции SymbolInfoInteger.

```
Print(EnumToString((ENUM_SYMBOL_TRADE_EXEC
UTION)
SymbolInfoInteger(Symbol(),SYMBOL_TRADE_EXEM
ODE)));
```

Для немедленного исполнения (Instant Execution), исполнение рыночного ордера осуществляется по цене, которую вы предлагаете брокеру.

Если брокер не может принять ордер по предложенным ценам, он предложит трейдеру новые цены исполнения, которые будут содержаться в структуре MqlTradeResult.

Для немедленного исполнения (Instant Execution), заполнение структуры MqlTradeRequest для ордера на покупку будет иметь следующий вид.

```
MqlTradeRequest mrequest;
ZeroMemory(mrequest);
MqlTick latest_price;

if(!SymbolInfoTick(_Symbol,latest_price))
    {
Alert("Ошибка получения последних котировок - ошибка:",GetLastError(),"!!");
     return;
    }
if(((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(),SYMBOL_TRAD
E_EXEMODE))==SYMBOL_TRADE_EXECUTION_INSTANT){

mrequest.action = TRADE_ACTION_DEAL;
mrequest.symbol = _Symbol;
mrequest.volume = Lot;
mrequest.price = NormalizeDouble(latest_price.ask,_Digits);
mrequest.sl = NormalizeDouble(latest_price.bid - 0.01,_Digits);
mrequest.tp = NormalizeDouble(latest_price.ask + 0.01,_Digits);
mrequest.deviation=10;
mrequest.type = ORDER_TYPE_BUY;
mrequest.type_filling = ORDER_FILLING_FOK;
}
```

Здесь с помощью функции SymbolInfoTick в структуру MqlTick получаются текущие цены символа для предложения их брокеру.

Далее для немедленного исполнения (Instant Execution) заполняются обязательные поля структуры MqlTradeRequest action, symbol, volume, price, sl, tp, deviation, type, type_filling.

На практике, максимально приемлемое отклонение от запрашиваемой цены deviation, задаваемое в пунктах, которое принимается брокером, не более 5 пунктов.

При сильном движении рынка, при поступлении ордера брокеру, если цена ушла на большее значение, произойдет так называемое «Перекотирование» (Requote) – брокер вернет цены, по которым может быть исполнен данный ордер.

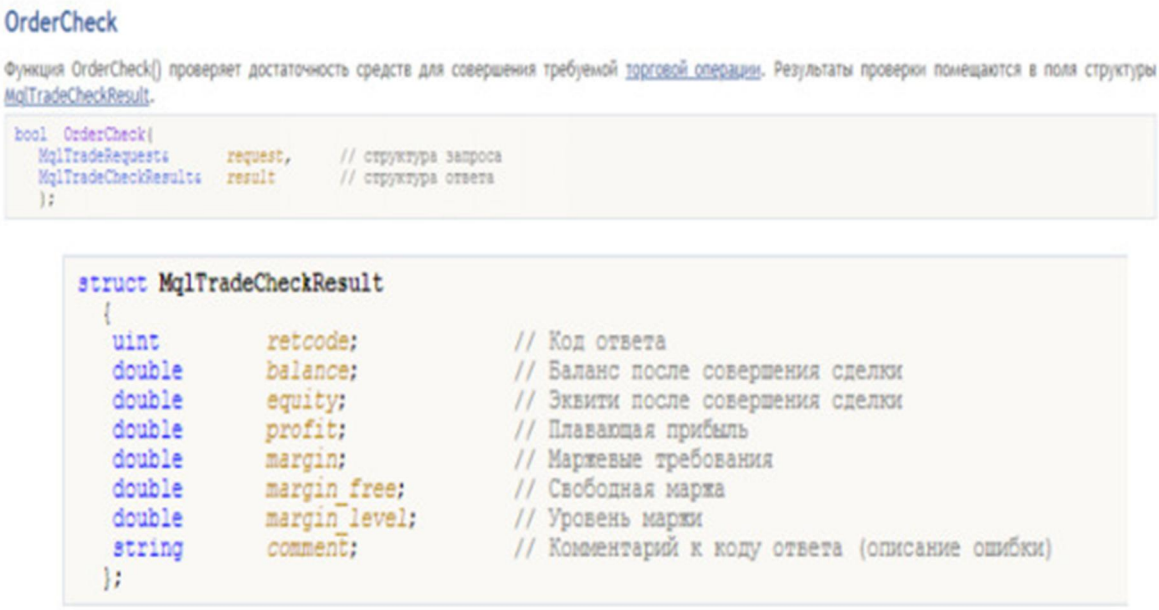
Функция NormalizeDouble здесь используется для округления цен до количества десятичных знаков после запятой, определяющего точность измерения цены символа текущего графика.

```
if(((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(),SYMBOL_TRAD
E_EXEMODE))==SYMBOL_TRADE_EXECUTION_INSTANT){

mrequest.action = TRADE_ACTION_DEAL;
mrequest.symbol = _Symbol;
mrequest.volume = Lot;
mrequest.price = NormalizeDouble(latest_price.ask,_Digits);
mrequest.sl = NormalizeDouble(latest_price.ask + 0.01,_Digits);
mrequest.tp = NormalizeDouble(latest_price.bid - 0.01,_Digits);
mrequest.deviation=10;
mrequest.type = ORDER_TYPE_SELL;
mrequest.type_filling = ORDER_FILLING_FOK;
}
```

Для ордера на продажу заполнение Instant Execution обязательных полей структуры MqlTradeRequest будет иметь следующий вид.

После заполнения полей структуры MqlTradeRequest рекомендуется проверить ее с помощью функции OrderCheck.



OrderCheck

Функция OrderCheck() проверяет достаточность средств для совершения требуемой торговой операции. Результаты проверки помещаются в поля структуры MqlTradeCheckResult.

```
bool  OrderCheck(
   MqlTradeRequest&      request,     // структура запроса
   MqlTradeCheckResult&  result       // структура ответа
   );
```

```
struct MqlTradeCheckResult
  {
   uint         retcode;             // Код ответа
   double       balance;             // Баланс после совершения сделки
   double       equity;              // Эквити после совершения сделки
   double       profit;              // Плавающая прибыль
   double       margin;              // Маржевые требования
   double       margin_free;         // Свободная маржа
   double       margin_level;        // Уровень маржи
   string       comment;             // Комментарий к коду ответа (описание ошибки)
  };
```

Результаты проверки будут содержаться в структуре MqlTradeCheckResult:

Функция OrderCheck возвращает true в случае успешной проверки структуры MqlTradeRequest, при этом код retcode Код ответа будет равен 0, в противном случае функция вернет false.

```
MqlTradeCheckResult check_result;
ZeroMemory(check_result);

if(!OrderCheck(mrequest,check_result))
   {
if(check_result.retcode==10014)Alert("Неправильный объем в запросе");
if(check_result.retcode==10015)Alert("Неправильная цена в запросе");
if(check_result.retcode==10016)Alert("Неправильные стопы в запросе");
if(check_result.retcode==10019)Alert("Нет достаточных денежных средств для выполнения
запроса");
     return;
   }
```

После проверки структуры MqlTradeRequest можно отсылать запрос на совершение торговой операции брокеру, используя функцию OrderSend.

```
//---------------------------------------------------------------------
MqlTradeRequest mrequest;
MqlTradeCheckResult check_result;
MqlTradeResult mresult;

MqlTick latest_price;
if(!SymbolInfoTick(_Symbol,latest_price))
    {
     Alert("Ошибка получения последних котировок - ошибка:",GetLastError(),"!!");
     return;
    }
if(TradeSignalBuy==true&&BuyOpened==false){
if(((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(),SYMBOL_TRAD
E_EXEMODE))==SYMBOL_TRADE_EXECUTION_INSTANT){
ZeroMemory(mrequest);
mrequest.action = TRADE_ACTION_DEAL;
mrequest.symbol = _Symbol;
mrequest.volume = Lot;
mrequest.price = NormalizeDouble(latest_price.ask,_Digits);
mrequest.sl = NormalizeDouble(latest_price.bid - 0.01,_Digits);
mrequest.tp = NormalizeDouble(latest_price.ask + 0.01,_Digits);
mrequest.deviation=10;
mrequest.type = ORDER_TYPE_BUY;
mrequest.type_filling = ORDER_FILLING_FOK;
```

Здесь мы получаем последние цены, затем проверяем флаг на покупку и флаг на открытую позицию.

Затем заполняем структуру торгового запроса MqlTradeRequest.

ORDER_FILLING_FOK означает, что ордер может быть исполнен исключительно в указанном объеме.

```
ZeroMemory(check_result);
ZeroMemory(mresult);
if(!OrderCheck(mrequest,check_result))
    {
if(check_result.retcode==10014)Alert("Неправильный объем в запросе");
if(check_result.retcode==10015)Alert("Неправильная цена в запросе");
if(check_result.retcode==10016)Alert("Неправильные стопы в запросе");
if(check_result.retcode==10019)Alert("Нет достаточных денежных средств для выполнения
запроса");
     return;
    }else{        if(OrderSend(mrequest,mresult)){
 if(mresult.retcode==10009 || mresult.retcode==10008) //запрос выполнен или ордер успешно
помещен
        {
Print("Price ", mresult.price);
        } else{        if(mresult.retcode==10004) //Реквота
{
Print("Requote bid ",mresult.bid);
Print("Requote ask ",mresult.ask);
}else{
Print("Retcode ",mresult.retcode);
}    }
}else{
Print("Retcode ",mresult.retcode);
```

Далее мы проверяем структуру MqlTradeRequest и отсылаем запрос на совершение торговой операции брокеру, используя функцию OrderSend.

```
if(TradeSignalSell==true&&SellOpened==false){

if(((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(),SYMBOL_TRAD
E_EXEMODE))==SYMBOL_TRADE_EXECUTION_INSTANT){

ZeroMemory(mrequest);
mrequest.action = TRADE_ACTION_DEAL;
mrequest.symbol = _Symbol;
mrequest.volume = Lot;
mrequest.price = NormalizeDouble(latest_price.ask,_Digits);
mrequest.sl = NormalizeDouble(latest_price.ask + 0.01,_Digits);
mrequest.tp = NormalizeDouble(latest_price.bid - 0.01,_Digits);
mrequest.deviation=10;
mrequest.type = ORDER_TYPE_SELL;
mrequest.type_filling = ORDER_FILLING_FOK;
```

Тоже самое мы делаем для открытия позиции на продажу.

Проверяем флаг на продажу и флаг на открытую позицию.

Затем заполняем структуру торгового запроса MqlTradeRequest.

```
ZeroMemory(check_result);
ZeroMemory(mresult);
if(!OrderCheck(mrequest,check_result))
    {
if(check_result.retcode==10014)Alert("Неправильный объем в запросе");
if(check_result.retcode==10015)Alert("Неправильная цена в запросе");
if(check_result.retcode==10016)Alert("Неправильные стопы в запросе");
if(check_result.retcode==10019)Alert("Нет достаточных денежных средств для выполнения запроса");
     return;
    }else{
 if(OrderSend(mrequest,mresult)){
 if(mresult.retcode==10009 || mresult.retcode==10008) //запрос выполнен или ордер успешно помещен
        {
Print("Price ", mresult.price);
        } else {
if(mresult.retcode==10004) //Реквота
{
Print("Requote bid ",mresult.bid);
Print("Requote ask ",mresult.ask);
}else{    Print("Retcode ",mresult.retcode);
}  }
}else{    Print("Retcode ",mresult.retcode);
}}}  }
```

Далее мы проверяем структуру MqlTradeRequest и отсылаем запрос на совершение торговой операции брокеру, используя функцию OrderSend.

Таким образом, запрос на совершение торговой операции отсылается, если есть сигнал на открытие позиции и при этом открытая позиция еще не существует.

После проверки OrderCheck производится повторная проверка структуры MqlTradeRequest в виде возвращаемого значения функции OrderSend.

Далее выполняется проверка кода результата операции структуры MqlTradeResult.

Исполнение ордера по запросу (Request Execution) я лично не встречал у брокеров.

Вместо немедленного исполнения (Instant Execution) брокер может предложить исполнение ордера по рынку (Market Execution) или биржевое исполнение (Exchange Execution).

В режиме исполнения по рынку (Market Execution) сделка совершается по цене, предложенной брокером, при этом реквоты отсутствуют.

В режиме биржевого исполнения (Exchange Execution) торговые операции якобы выводятся во внешнюю торговую систему и сделки выполняются по текущим рыночным ценам, при этом реквоты также отсутствуют.

```
//--------------------------------------------------------------------
if(((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(),SYMBOL_TRAD
E_EXEMODE))==SYMBOL_TRADE_EXECUTION_EXCHANGE){
//или SYMBOL_TRADE_EXECUTION_MARKET

ZeroMemory(mrequest);
mrequest.action = TRADE_ACTION_DEAL;
mrequest.symbol = _Symbol;
mrequest.volume = Lot;
mrequest.type = ORDER_TYPE_BUY;
mrequest.type_filling = ORDER_FILLING_FOK;
```

При исполнении по рынку (Market Execution) или биржевом исполнении (Exchange Execution) обязательными для заполнения являются поля структуры MqlTradeRequest action, symbol, volume, type, type_filling.

```
ZeroMemory(check_result);
ZeroMemory(mresult);
if(!OrderCheck(mrequest,check_result))
    {
if(check_result.retcode==10014)Alert("Неправильный объем в запросе");
if(check_result.retcode==10019)Alert("Нет достаточных денежных средств для выполнения запроса");
     return;
    }else{
 if(OrderSend(mrequest,mresult)){
 if(mresult.retcode==10009 || mresult.retcode==10008) //запрос выполнен или ордер успешно помещен
        {
//------------------------
ZeroMemory(mrequest);
mrequest.action = TRADE_ACTION_SLTP;
mrequest.symbol = _Symbol;
mrequest.sl = NormalizeDouble(mresult.price - 0.01,_Digits);
mrequest.tp = NormalizeDouble(mresult.price + 0.01,_Digits);
ZeroMemory(check_result);
ZeroMemory(mresult);
```

После заполнения структуры MqlTradeRequest мы ее проверяем и посылаем запрос брокеру.

Далее мы формируем новый запрос, в котором устанавливаем значения Stop Loss и Take Profit у открытой позиции, исходя из цен, полученным от брокера.

```
if(!OrderCheck(mrequest,check_result))
    {
if(check_result.retcode==10015)Alert("Неправильная цена в запросе");
if(check_result.retcode==10016)Alert("Неправильные стопы в запросе");
     return;
    }else{
 if(OrderSend(mrequest,mresult)){
 if(mresult.retcode==10009 || mresult.retcode==10008) //запрос выполнен или ордер
успешно помещен
        {
Print("SL ", mrequest.sl, "TP ",mrequest.tp);
        }
       else {    Print("Retcode ",mresult.retcode);
        }
}else{          Print("Retcode ",mresult.retcode);
}} }else{       Print("Retcode ",mresult.retcode);
        }
}else{          Print("Retcode ",mresult.retcode);
}} }
```

После заполнения новой структуры MqlTradeRequest мы ее проверяем и посылаем новый запрос брокеру.

```
//------------------------------------------------------------------
if(((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(),SYMBOL_TRAD
E_EXEMODE))==SYMBOL_TRADE_EXECUTION_EXCHANGE){
//или SYMBOL_TRADE_EXECUTION_MARKET

ZeroMemory(mrequest);
mrequest.action = TRADE_ACTION_DEAL;
mrequest.symbol = _Symbol;
mrequest.volume = Lot;
mrequest.type = ORDER_TYPE_SELL;
mrequest.type_filling = ORDER_FILLING_FOK;
```

Тоже самое делаем при открытии позиции на продажу.

Заполняем структуру MqlTradeRequest.

```
ZeroMemory(check_result);
ZeroMemory(mresult);

if(!OrderCheck(mrequest,check_result))
   {
if(check_result.retcode==10014)Alert("Неправильный объем в запросе");
if(check_result.retcode==10019)Alert("Нет достаточных денежных средств для выполнения
запроса");
     return;
   }else{
 if(OrderSend(mrequest,mresult)){
 if(mresult.retcode==10009 || mresult.retcode==10008) //запрос выполнен или ордер успешно
помещен
        {

//----------------------
ZeroMemory(mrequest);
mrequest.action = TRADE_ACTION_SLTP;
mrequest.symbol = _Symbol;
mrequest.tp = NormalizeDouble(mresult.price - 0.01,_Digits);
mrequest.sl = NormalizeDouble(mresult.price + 0.01,_Digits);
ZeroMemory(check_result);
ZeroMemory(mresult);
```

После заполнения структуры MqlTradeRequest мы ее проверяем и посылаем запрос брокеру.

Далее мы формируем новый запрос, в котором устанавливаем значения Stop Loss и Take Profit у открытой позиции, исходя из цен, полученным

от брокера.

```
if(!OrderCheck(mrequest,check_result))
   {
if(check_result.retcode==10015)Alert("Неправильная цена в запросе");
if(check_result.retcode==10016)Alert("Неправильные стопы в запросе");
    return;
   }else{
 if(OrderSend(mrequest,mresult)){
 if(mresult.retcode==10009 || mresult.retcode==10008) //запрос выполнен или ордер успешно помещен
       {
Print("SL ", mrequest.sl, "TP ",mrequest.tp);
       }else{      Print("Retcode ",mresult.retcode);
       }
}else{              Print("Retcode ",mresult.retcode);
}} }else{           Print("Retcode ",mresult.retcode);
       }
}else{              Print("Retcode ",mresult.retcode);
}} }
```

После заполнения новой структуры MqlTradeRequest мы ее проверяем и посылаем новый запрос брокеру.

Таким образом, здесь позиция открывается без определения StopLoss и TakeProfit, а затем, в случае успешного выполнения запроса, размещается торговый приказ на модификацию уровней StopLoss и TakeProfit.

ORDER_TYPE_BUY_LIMIT – отложенный ордер на покупку, при этом текущие цены выше цены ордера.

ORDER_TYPE_SELL_LIMIT – отложенный ордер на продажу, при этом текущие цены ниже цены ордера.

ORDER_TYPE_BUY_STOP – отложенный ордер на покупку, при этом текущие цены ниже цены ордера.

ORDER_TYPE_SELL_STOP – отложенный ордер на продажу, при этом текущие цены выше цены ордера.

ORDER_TYPE_BUY_STOP_LIMIT – отложенное выставление отложенного ордера типа Buy Limit для торговли на откате, при этом текущие цены ниже цены ордера.

ORDER_TYPE_SELL_STOP_LIMIT – отложенное выставление отложенного ордера типа Sell Limit для торговли на откате, при этом текущие цены выше цены ордера.

Для установки отложенного ордера на покупку или продажу (TRADE_ACTION_PENDING), требуется указание 11 полей структуры MqlTradeRequest: action, symbol, volume, price, stoplimit, sl, tp, type, type_filling, type_time, expiration.

При этом поле type может принимать следующие значения.

– ORDER_TYPE_BUY_LIMIT – отложенный ордер на покупку, при этом текущие цены выше цены ордера.

– ORDER_TYPE_SELL_LIMIT – отложенный ордер на продажу, при этом текущие цены ниже цены ордера.

– ORDER_TYPE_BUY_STOP – отложенный ордер на покупку, при этом текущие цены ниже цены ордера.

– ORDER_TYPE_SELL_STOP – отложенный ордер на продажу, при этом текущие цены выше цены ордера.

– ORDER_TYPE_BUY_STOP_LIMIT – отложенное выставление отложенного ордера типа Buy Limit для торговле на откате, при этом текущие цены ниже цены ордера.

– ORDER_TYPE_SELL_STOP_LIMIT – отложенное выставление отложенного ордера типа Sell Limit для торговли на откате, при этом текущие цены выше цены ордера.

Для изменения параметров отложенного ордера (TRADE_ACTION_MODIFY), требуется указание 7 полей структуры MqlTradeRequest: action, order, price, sl, tp, type_time, expiration.

При этом значение поля order берется из структуры MqlTradeResult результата выставления ордера.

Для удаления отложенного ордера (TRADE_ACTION_REMOVE), требуется указание 2 полей структуры MqlTradeRequest: action и order.

Пример создания эксперта

В качестве основы советника возьмем следующий код.

```mql5
input double   Lot=1;
input int      EA_Magic=1000;
input double spreadLevel=5.0;
input double StopLoss=0.01;
input double Profit=0.01;

bool flagStopLoss=false;

int OnCheckTradeInit(){
//Проверка запуска эксперта на реальном счете
if((ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL){
  int mb=MessageBox("Запустить советник на реальном счете?","Message Box",MB_YESNO|MB_ICONQUESTION);
  if(mb==IDNO) return(0);
 }
//-----------------------------------------------
//Проверки:запрещена торговля в случае подключения к счету в режиме инвестора,
//отсутствия соединения к серверу, запрета торговли на стороне сервера, если счет отправлен в архив
//брокер запрещает автоматическую торговлю

if(!TerminalInfoInteger(TERMINAL_CONNECTED)){
Alert("No connection to the trade server");
return(0);
}else{
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED)){
Alert("Trade for this account is prohibited");
return(0);
  }
 }
```

В этом коде, мы сначала определяем объем торговли в лотах, максимально допустимый спред брокера, при котором мы согласны торговать, и значения стоплосса и тейкпрофита для нашей торговли, которые потом можно оптимизировать.

Также мы определяем флаг flagStopLoss, для того чтобы прекратить торговлю, если поймаем стоплосс.

И здесь общие проверки функции OnInit выделены в отдельную функцию OnCheckTradeInit, а общие проверки функции OnTick выделены в отдельную функцию OnCheckTradeTick.

```mql5
int OnCheckTradeInit(){
//Проверка запуска эксперта на реальном счете
if((ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL){
  int mb=MessageBox("Запустить советник на реальном счете?","Message Box",MB_YESNO|MB_ICONQUESTION);
  if(mb==IDNO) return(0);
 }
//Проверки:
//Отсутствие соединения к серверу, запрета торговли на стороне сервера
//Брокер запрещает автоматическую торговлю

if(!TerminalInfoInteger(TERMINAL_CONNECTED)){
Alert("No connection to the trade server");
return(0);
}
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED)){
Alert("Trade for this account is prohibited");
return(0);
}
if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT)){
Alert("Trade with the help of experts for the account is prohibited");
return(0);
}
//Проверить корректность объема, с которым мы собираемся выйти на рынок
   if(Lot<SymbolInfoDouble(Symbol(),SYMBOL_VOLUME_MIN)||Lot>SymbolInfoDouble(Symbol(),SYMBOL_VOLUME_MAX)){
 Alert("Lot is not correct!!!");
      return(0);
}
   return(INIT_SUCCEEDED);
}
```

В функции OnCheckTradeInit мы спрашиваем трейдера разрешение на запуск эксперта на реальном счете, а затем проверяем соединение к серверу, не запрещена ли торговля на стороне сервера, и не запретил ли брокер автоматическую торговлю.

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
return(OnCheckTradeInit());
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
  }
```

Соответственно в функции OnInit, мы просто вызываем нашу функцию OnCheckTradeInit.

```
int OnCheckTradeTick(){
//Проверка отсутствия соединения к серверу
if(!TerminalInfoInteger(TERMINAL_CONNECTED)){
Alert("No connection to the trade server");
return(0);
}
//Включена ли кнопка авто-торговли в клиентском терминале
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED)){
Alert("Разрешение на автоматическую торговлю выключено!");
return(0);
}
//Разрешение на торговлю с помощью эксперта отключено в общих свойствах самого эксперта
if(!MQLInfoInteger(MQL_TRADE_ALLOWED)){
Alert("Автоматическая торговля запрещена в свойствах эксперта ",__FILE__);
return(0);
}
//Уровень залоговых средств, при котором требуется пополнение счета
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT){
if(AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)!=0&&AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL)){
Alert("Margin Call!!!");
return(0);
}}
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY){
if(AccountInfoDouble(ACCOUNT_EQUITY)<=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL)){
Alert("Margin Call!!!");
return(0);
}}
```

В функции OnCheckTradeTick, общих проверок функции OnTick, мы проверяем соединение к серверу, включена ли кнопка авто-торговли в клиентском терминале, включено ли разрешение на торговлю с помощью эксперта в общих свойствах самого эксперта, и наступление события Margin Call.

```
//Уровень залоговых средств, при достижении которого происходит принудительное закрытие самой убыточной позиции
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT){
if(AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)!=0&&AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)){
Alert("Stop Out!!!");
return(0);
}}
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY){
if(AccountInfoDouble(ACCOUNT_EQUITY)<=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)){
Alert("Stop Out!!!");
return(0);
}}
//Проверка размера свободных средств на счете, доступных для открытия позиции
 double margin;
 MqlTick last_tick;
 ResetLastError();
 if(SymbolInfoTick(Symbol(),last_tick))
     {
      if(OrderCalcMargin(ORDER_TYPE_BUY,Symbol(),Lot,last_tick.ask,margin))
        {
     if(margin>AccountInfoDouble(ACCOUNT_MARGIN_FREE)){
      Alert("Not enough money in the account!");
      return(0);
     }
        }
     }
   else
     {
      Print(GetLastError());
     }
```

Далее мы проверяем наступление события Stop Out и достаточно ли свободных средств на счете для открытия позиции.

```
//Контроль над спредом брокера
double _spread=
SymbolInfoInteger(Symbol(),SYMBOL_SPREAD)*MathPow(10,-SymbolInfoInteger(Symbol(),SYMBOL_DIGITS))/MathPow(10,-4);
 if(_spread>spreadLevel){
 Alert("Слишком большой спред!");
 return(0);
 }
//Проверка ограничений на торговые операции по символу, установленные брокером
if((ENUM_SYMBOL_TRADE_MODE)SymbolInfoInteger(Symbol(),SYMBOL_TRADE_MODE)!=SYMBOL_TRADE_MODE_FULL){
Alert("Установлены ограничения на торговые операции");
return(0);
}
//Достаточно ли баров в истории для расчета советника
if(Bars(Symbol(), 0)<100)
     {
      Alert("In the chart little bars, Expert will not work!!");
      return(0);
     }

     return(1);
}
```

И затем, мы осуществляем контроль над спредом брокера, проверяем ограничения на торговые операции по символу, установленные брокером, и проверяем, достаточно ли баров в истории для расчета советника.

```
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
if(!OnCheckTradeTick()){
return;
}
//Ограничить вычисления советника по появлению нового бара на графике
static datetime last_time;
datetime last_bar_time=(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_LASTBAR_DATE);
if(last_time!=last_bar_time)
{
last_time=last_bar_time;
}else{
return;
}
//Ограничить вычисления советника по flagStopLoss
static datetime last_time_daily;
datetime last_bar_time_daily=(datetime)SeriesInfoInteger(Symbol(),PERIOD_D1,SERIES_LASTBAR_DATE);
if(last_time_daily!=last_bar_time_daily)
{
last_time_daily=last_bar_time_daily;
flagStopLoss=false;
}
if(flagStopLoss==true)return;
```

Соответственно в функции OnTick, мы сначала вызываем функцию OnCheckTradeTick для осуществления всех проверок.

Затем мы ограничиваем работу эксперта по появлению нового бара на графике символа, а не при каждом тике.

Для этого мы используем статическую переменную last_time, в которую записываем время появления последнего бара.

Далее мы ограничиваем дневную торговлю при получении стоплосса.

Этот шаг конечно же является необязательным и зависит от стратегии трейдера.

Мы приводим его исключительно для демонстрации работы функции OnTrade.

```
169 //Проверка наличия открытой позиции, чтобы не пытаться открыть ее заново
170    bool BuyOpened=false;
171    bool SellOpened=false;
172    if(PositionSelect(_Symbol)==true)
173      {
174       if(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE_BUY)
175         {
176          BuyOpened=true;
177         }
178       else if(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE_SELL)
179         {
180          SellOpened=true;
181         }
182      }
183 //Для вычисления сигналов торговой системы требуются исторические данные символа
184 MqlRates mrate[];
185 ResetLastError();
186 if(CopyRates(Symbol(),Period(),0,3,mrate)<0)
187      {
188 Print(GetLastError());
189       return;
190      }
191
192 ArraySetAsSeries(mrate,true);
193
194 bool TradeSignalBuy=false;
195 bool TradeSignalSell=false;
```

Затем мы проверяем наличие открытой позиции, чтобы не пытаться открыть ее заново, используя флаги BuyOpened и SellOpened.

И получаем цены символа для вычисления сигналов торговой системы.

Цены записываем в структуру MqlRates.

И советник будет отправлять ордера на покупку и продажу при установке флагов TradeSignalBuy и TradeSignalSell в значение true.

Установку значений флагов TradeSignalBuy и TradeSignalSell должна осуществлять торговая система.

```
MqlTradeRequest mrequest;
MqlTradeCheckResult check_result;
MqlTradeResult mresult;

MqlTick latest_price;
if(!SymbolInfoTick(_Symbol,latest_price))
     {
      Alert("Ошибка получения последних котировок - ошибка:",GetLastError(),"!!");
      return;
     }
if(TradeSignalBuy==true&&BuyOpened==false){

if(((ENUM_SYMBOL_TRADE_EXECUTION)SymbolInfoInteger(Symbol(),SYMBOL_TRADE_EXEMODE))==SYMBOL_TRADE_EXECUTION_INSTANT){
ZeroMemory(mrequest);
mrequest.action = TRADE_ACTION_DEAL;
mrequest.symbol = _Symbol;
mrequest.volume = Lot;
mrequest.price = NormalizeDouble(latest_price.ask,_Digits);
mrequest.sl = NormalizeDouble(latest_price.bid - StopLoss,_Digits);
mrequest.tp = NormalizeDouble(latest_price.ask + Profit,_Digits);
mrequest.deviation=10;
mrequest.type = ORDER_TYPE_BUY;
mrequest.type_filling = ORDER_FILLING_FOK;

ZeroMemory(check_result);
ZeroMemory(mresult);
if(!OrderCheck(mrequest,check_result))
     {
     if(check_result.retcode==10014)Alert("Неправильный объем в запросе");
     if(check_result.retcode==10015)Alert("Неправильная цена в запросе");
     if(check_result.retcode==10016)Alert("Неправильные стопы в запросе");
     if(check_result.retcode==10019)Alert("Нет достаточных денежных средств для выполнения запроса");
      return;
```

Далее, при получении сигналов от торговой системы, мы отправляем ордера на покупку или продажу.

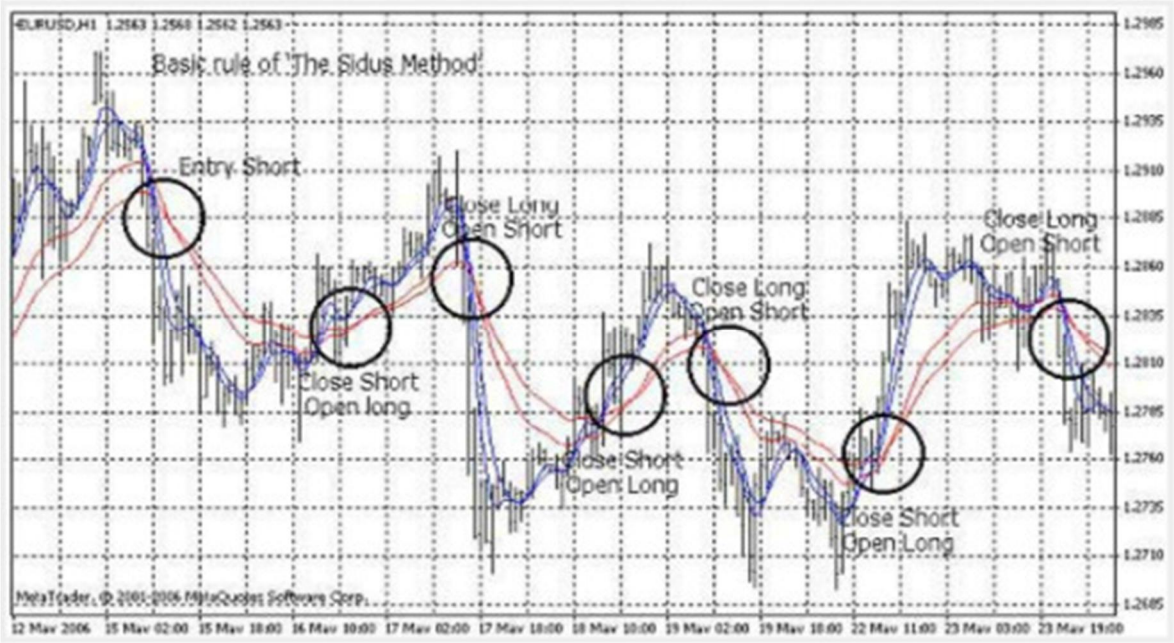
```
void OnTrade()
  {
static int _deals;
ulong _ticket=0;

if(HistorySelect(0,TimeCurrent()))
  {
 int  i=HistoryDealsTotal()-1;

   if(_deals!=i) {
   _deals=i;
   } else { return; }

   if((_ticket=HistoryDealGetTicket(i))>0)
     {
      string _comment=HistoryDealGetString(_ticket,DEAL_COMMENT);
      if(StringFind(_comment,"sl",0)!=-1) {
      flagStopLoss=true;
      }
     }
  }
  }
```

И наконец, в функции OnTrade мы обрабатываем событие стоплосса и устанавливаем флаг flagStopLoss в значение true.



В качестве торговой системы возьмем «Метод Сидуса».

Здесь берется временной интервал – H1 – часы.

И экспоненциальные скользящие средние (Exponential Moving Average): 18 EMA и 28 EMA;

А также Weighted Moving Average – 5WMA и 8 WMA.

И торговые сигналы на вход в рынок по Методу Сидуса:

Открытие позиции на покупку: 5WMA и 8WMA скользящие средние пересекают туннель из 18EMA и 28EMA снизу вверх.

Открытие позиции на продажу: 5WMA и 8 WMA скользящие средние пересекают туннель из 18 EMA и 28 EMA сверху вниз.

Торговые сигналы на выход из рынка по Методу Сидуса:

На покупку: цена на графике достигла вершины и 5 WMA как бы «ныряет» под 8 WMA скользящую среднюю. Следует закрыть открытую торговую позицию.

На продажу: цена на графике достигла дна и скользящая средняя 5 WMA как бы «прыгает» над 8WMA. Следует закрыть торговую позицию.

```
int     handleIMA18;
double    MA18Buffer[];
int     handleIMA28;
double    MA28Buffer[];
int     handleIWMA5;
double    WMA5Buffer[];
int     handleIWMA8;
double    WMA8Buffer[];
```

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
handleIMA18=iMA(_Symbol,PERIOD_H1,18,0,MODE_EMA,PRICE_CLOSE);
handleIMA28=iMA(_Symbol,PERIOD_H1,28,0,MODE_EMA,PRICE_CLOSE);
handleIWMA5=iMA(_Symbol,PERIOD_H1,5,0,MODE_LWMA,PRICE_CLOSE);
handleIWMA8=iMA(_Symbol,PERIOD_H1,8,0,MODE_LWMA,PRICE_CLOSE);

return(OnCheckTradeInit());
  }
```

Теперь дополним код советника, реализацией Метода Сидуса.
Объявим хэндля используемых индикаторов и их буферы.
И в функции OnInit получим эти хэндлы.

```
bool TradeSignalBuy=false;
bool TradeSignalSell=false;

TradeSignalBuy=OnTradeSignalBuy();
TradeSignalSell=OnTradeSignalSell();

bool TradeSignalBuyStop=false;
bool TradeSignalSellStop=false;


TradeSignalBuyStop=OnTradeSignalBuyStop(mrate);
TradeSignalSellStop=OnTradeSignalSellStop(mrate);
```

В функции OnTick, мы вводим новые функции OnTradeSignalBuy
и OnTradeSignalSell для вычисления сигналов на покупку и продажу.

И функции OnTradeSignalBuyStop и OnTradeSignalSellStop для вычисления сигналов на закрытие позиций на покупку и продажу.

```
#include <Trade\PositionInfo.mqh>
#include <Trade\Trade.mqh>
CPositionInfo  m_position;                   // trade position object
CTrade         m_trade;                      // trading object

//---------------------------------------------------------
if(TradeSignalSellStop==true&&SellOpened==true){
      for(int i=PositionsTotal()-1;i>=0;i--) // returns the number of current positions
      if(m_position.SelectByIndex(i))     // selects the position by index for further access to its properties
      {
           ENUM_POSITION_TYPE type = m_position.PositionType();
           if(type==POSITION_TYPE_SELL)
           m_trade.PositionClose(m_position.Ticket()); // close a position by the specified symbol
      }
}
//---------------------------------------------------------------------------------------------------------------
if(TradeSignalBuyStop==true&&BuyOpened==true){
      for(int i=PositionsTotal()-1;i>=0;i--) // returns the number of current positions
      if(m_position.SelectByIndex(i))     // selects the position by index for further access to its properties
      {
           ENUM_POSITION_TYPE type = m_position.PositionType();
           if(type==POSITION_TYPE_BUY)
           m_trade.PositionClose(m_position.Ticket()); // close a position by the specified symbol

       }
}
```

Таким образом, мы должны дополнить код функции OnTick закрытием позиции на покупку и продажу.

В самом начале появления MQL5 закрытие позиции осуществлялось путем отправки противоположного ордера с тем же объемом, то есть закрытие позиции на покупку делалось путем отправки ордера на продажу с тем же объемом, и наоборот, для закрытия позиции на продажу.

То есть платформа MetaTrader 5 изначально создавалась для биржевой торговли с неттинговым учетом позиций.

При неттинговом учете по одному финансовому инструменту можно иметь только одну позицию, поэтому все дальнейшие операции по нему ведут к изменению объема, закрытию или развороту существующей позиции.

Но чтобы расширить возможности трейдеров, в платформу была добавлена вторая система учета – хеджинг.

Теперь по инструменту можно иметь множество позиций, в том числе – разнонаправленных.

Это позволяет реализовывать торговые стратегии с так называемым локированием – если цена пошла против трейдера, он имеет возможность открыть позицию в противоположном направлении.

Поэтому для надежного закрытия позиции мы используем библиотечные классы CPositionInfo и CTrade.

Поэтому включим в код файлы этих классов и создадим их экземпляры.

И в функции OnTick, мы в цикле проверяем все открытые позиции, и с помощью метода PositionClose класса CTrade закрываем позиции определенного типа.

```
input int numberBarOpenPosition=5;
input int numberBarStopPosition=5;
```

```
bool OnTradeSignalBuy(){
bool flagBuy=false;
 if(CopyBuffer(handleIMA18,0,0,numberBarOpenPosition,MA18Buffer)<0)
     {
      return false;
     }
 if(CopyBuffer(handleIMA28,0,0,numberBarOpenPosition,MA28Buffer)<0)
     {
      return false;
     }
 if(CopyBuffer(handleIWMA5,0,0,numberBarOpenPosition,WMA5Buffer)<0)
     {
      return false;
     }
 if(CopyBuffer(handleIWMA8,0,0,numberBarOpenPosition,WMA8Buffer)<0)
     {
      return false;
     }
 ArraySetAsSeries(MA18Buffer,true);
 ArraySetAsSeries(MA28Buffer,true);
 ArraySetAsSeries(WMA5Buffer,true);
 ArraySetAsSeries(WMA8Buffer,true);
```

И конечно же мы должны определить функции сигналов торговой системы, так как в методе OnTick для получения сигналов на продажу или покупку вызываются функции OnTradeSignalBuy, OnTradeSignalSell, OnTradeSignalBuyStop, OnTradeSignalSellStop.

Но сначала, перед функциями обратного вызова, мы объявляем входные параметры numberBarOpenPosition – количество баров, на которых будет проверяться пересечение 5WMA и 8 WMA туннеля из 18 EMA и 28 EMA, и numberBarStopPosition – количество баров, на которых будет проверяться пересечение 5WMA и 8 WMA и достижения ценой вершины или дна.

В функции OnTradeSignalBuy и функции OnTradeSignalSell с помощью хэндлов индикаторов заполняются динамические массивы значений индикаторов.

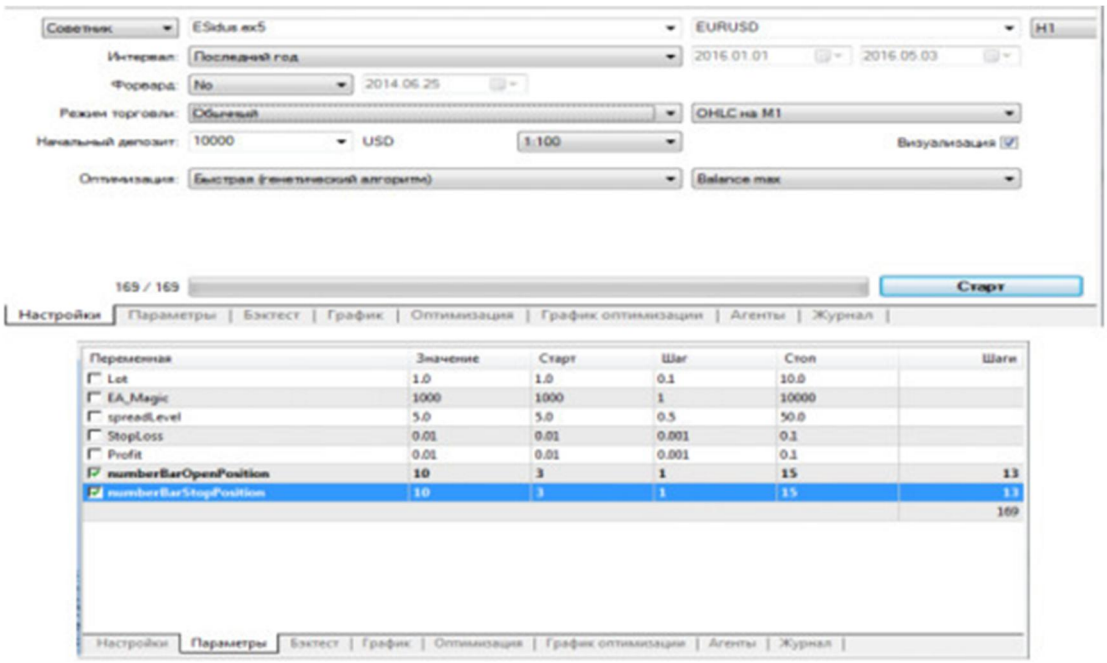
```
526 bool flagCross1=false;
527 bool flagCross2=false;
528 bool flagCross=false;
529
530 if(WMA5Buffer[1]>MA18Buffer[1]&&WMA5Buffer[1]>MA28Buffer[1]
531 &&WMA8Buffer[1]>MA18Buffer[1]&&WMA8Buffer[1]>MA28Buffer[1]){
532 for (int i=2;i<numberBarOpenPosition;i++){
533 if(WMA5Buffer[i]<MA18Buffer[i]&&WMA5Buffer[i]<MA28Buffer[i]){
534 flagCross1=true;
535 }
536 if(WMA8Buffer[i]<MA18Buffer[i]&&WMA8Buffer[i]<MA28Buffer[i]){
537 flagCross2=true;
538 }
539 }
540 if(flagCross1==true&&flagCross2==true){
541 flagCross=true;
542 }
543 }
544 flagBuy=flagCross;
545 return flagBuy;
546 }
```

И на количестве баров numberBarOpenPosition проверяется пресечение 5WMA и 8 WMA туннеля из 18 EMA и 28 EMA.

```
548 bool OnTradeSignalBuyStop(MqlRates& mrate[]){
549 bool flagBuyStop=false;
550  if(CopyBuffer(handleIWMA5,0,0,numberBarStopPosition,WMA5Buffer)<0)
551      {
552       return false;
553      }
554  if(CopyBuffer(handleIWMA8,0,0,numberBarStopPosition,WMA8Buffer)<0)
555      {
556       return false;
557      }
558  ArraySetAsSeries(WMA5Buffer,true);
559  ArraySetAsSeries(WMA8Buffer,true);
560  bool flagCross=false;
561  if(WMA5Buffer[1]<WMA8Buffer[1]){
562  for (int i=2;i<numberBarStopPosition;i++){
563 if(WMA5Buffer[i]>WMA8Buffer[i]){
564  flagCross=true;
565  } }  }
566  double max=mrate[1].high;
567  for (int i=1;i<numberBarStopPosition;i++){
568  if(mrate[i].high>max)max=mrate[i].high;
569  }
570  if(flagCross==true&&mrate[1].high<=max&&mrate[numberBarStopPosition-1].high<=max){
571  flagBuyStop=true;
572  }
573 return flagBuyStop;
574 }
```

В функции OnTradeSignalBuyStop и функции OnTradeSignalSellStop с помощью хэндлов индикаторов заполняются динамические массивы значений индикаторов и на количестве баров numberBarStopPosition проверяется пресечение 5WMA и 8 WMA и достижения ценой вершины или дна.

После компиляции советника в клиентском терминале нажмем правой кнопкой мышки на советнике и выберем Тестировать.



Попробуем оптимизировать параметры numberBarOpenPosition и numberBarStopPosition.

Результат	Прибыль	Всего трейдов	numberBarOpenPosition	numberBarStopPosition
12492.08	2492.08	28	11	4
12492.08	2492.08	28	11	3
12466.97	2466.97	30	12	4
12466.97	2466.97	30	12	3
12350.72	2350.72	33	14	4
12350.72	2350.72	33	13	4
12350.72	2350.72	33	14	3
12350.72	2350.72	33	13	3
12312.72	2312.72	34	15	4
12312.72	2312.72	34	15	3
10371.67	371.67	19	9	15
10371.67	371.67	19	9	14
10371.67	371.67	19	9	13

В результате оптимизации получим максимальный профит при значении numberBarOpenPosition = 11, и при значении numberBarStopPosition=4.

При изменении параметра цены индикаторов на PRICE_WEIGHTED показатель прибыли улучшается.

MathAbs(WMA5Buffer[1]-WMA5Buffer[numberBarStopPosition-1])<0.001

Условие достижения рынком дна или вершины можно заменить на горизонтальность линии EMA.

MathAbs (WMA5Buffer [1] -WMA5Buffer [numberBarStopPosition-1]) <0.001

И можно убрать действие сигналов TradeSignalBuyStop и TradeSignalSellStop и работать только на достижение Take Profit или Stop Loss.

Пример создания эксперта с использованием ООП

В предыдущем примере советника выделим функции OnCheckTradeInit, OnCheckTradeTick, OnTradeSignalBuy, OnTradeSignalBuyStop, OnTradeSignalSell, OnTradeSignalSellStop, а также код открытия и закрытия позиции в отдельные классы.

```mql5
class CheckTrade
  {
private:

public:
                     CheckTrade();
                    ~CheckTrade();
int                  OnCheckTradeInit(double   lot);
int                  OnCheckTradeTick(double   lot, double spread);
  };
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
CheckTrade::CheckTrade()
  {
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
CheckTrade::~CheckTrade()
  {
  }
```

Проверочные функции OnCheckTradeInit и OnCheckTradeTick выделим в класс CheckTrade.

В этом классе мы объявляем два метода OnCheckTradeInit и OnCheckTradeTick.

```mql5
//+------------------------------------------------------------------+
int CheckTrade::OnCheckTradeInit(double   lot){
//Проверка запуска эксперта на реальном счете
if((ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL){
  int mb=MessageBox("Запустить советник на реальном счете?","Message Box",MB_YESNO|MB_ICONQUESTION);
  if(mb==IDNO) return(0);
 }
//Проверки:
//отсутствие соединения к серверу, запрет торговли на стороне сервера
//брокер запрещает автоматическую торговлю
if(!TerminalInfoInteger(TERMINAL_CONNECTED)){
Alert("No connection to the trade server");
return(0);
}else{
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED)){
Alert("Trade for this account is prohibited");
return(0);
  }
 }
  if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT)){
      Alert("Trade with the help of experts for the account is prohibited");
   return(0);
  }
//Проверить корректность объема, с которым мы собираемся выйти на рынок
   if(lot<SymbolInfoDouble(Symbol(),SYMBOL_VOLUME_MIN)||lot>SymbolInfoDouble(Symbol(),SYMBOL_VOLUME_MAX)){
 Alert("Lot is not correct!!!");
      return(0);
}
   return(INIT_SUCCEEDED);

}
```

В методе OnCheckTradeInit класса мы запрашиваем трейдера для запуска эксперта на реальном счете.

Затем мы проверяем соединение к серверу, отсутствие запрета торговли на стороне сервера, и отсутствие запрета брокером автоматической торговли.

И наконец, мы проверяем корректность объема, с которым мы собираемся выйти на рынок.

```
int CheckTrade::OnCheckTradeTick(double   lot,double spread){
//Проверка отсутствия соединения к серверу
if(!TerminalInfoInteger(TERMINAL_CONNECTED)){
Alert("No connection to the trade server");
return(0);
}
//Включена ли кнопка авто-торговли в клиентском терминале
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED)){
Alert("Разрешение на автоматическую торговлю выключено!");
return(0);
}
//Разрешение на торговлю с помощью эксперта отключено в общих свойствах самого эксперта
if(!MQLInfoInteger(MQL_TRADE_ALLOWED)){
Alert("Автоматическая торговля запрещена в свойствах эксперта ",__FILE__);
return(0);
}
//Уровень залоговых средств, при котором требуется пополнение счета
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT){
if(AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)!=0&&AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL)){
Alert("Margin Call!!!");
return(0);
}}
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY){
if(AccountInfoDouble(ACCOUNT_EQUITY)<=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL)){
Alert("Margin Call!!!");
return(0);
}}
```

В методе OnCheckTradeTick мы проверяем соединение к серверу, включена ли кнопка авто-торговли в клиентском терминале, есть ли разрешение на торговлю с помощью эксперта в общих свойствах самого эксперта, и наступление события Margin Call.

```
//Уровень залоговых средств, при достижении которого происходит принудительное закрытие самой убыточной позиции
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_PERCENT){
if(AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)!=0&&AccountInfoDouble(ACCOUNT_MARGIN_LEVEL)
<=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)){
Alert("Stop Out!!!");
return(0);
}}
if((ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_MARGIN_SO_MODE)==ACCOUNT_STOPOUT_MODE_MONEY){
if(AccountInfoDouble(ACCOUNT_EQUITY)<=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO)){
Alert("Stop Out!!!");
return(0);
}}
//Проверка размера свободных средств на счете, доступных для открытия позиции
 double margin;
 MqlTick last_tick;
 ResetLastError();
 if(SymbolInfoTick(Symbol(),last_tick))
     {
      if(OrderCalcMargin(ORDER_TYPE_BUY,Symbol(),lot,last_tick.ask,margin))
        {
     if(margin>AccountInfoDouble(ACCOUNT_MARGIN_FREE)){
      Alert("Not enough money in the account!");
      return(0);
     }}
     }else{
      Print(GetLastError());
     }
```

Далее мы проверяем наступление события Stop Out и проверяем размер свободных средств на счете, доступных для открытия позиции.

```
//Контроль над спредом брокера
double _spread=
SymbolInfoInteger(Symbol(),SYMBOL_SPREAD)*MathPow(10,-SymbolInfoInteger(Symbol(),SYMBOL_DIGITS))/MathPow(10,-4);
 if(_spread>spread){
 Alert("Слишком большой спред!");
 return(0);
 }
//Проверка ограничений на торговые операции по символу, установленные брокером
if((ENUM_SYMBOL_TRADE_MODE)SymbolInfoInteger(Symbol(),SYMBOL_TRADE_MODE)!=SYMBOL_TRADE_MODE_FULL){
Alert("Установлены ограничения на торговые операции");
return(0);
}
//Достаточно ли баров в истории для расчета советника
if(Bars(Symbol(), 0)<100)
     {
      Alert("In the chart little bars, Expert will not work!!");
      return(0);
     }

     return(1);
}
```

И затем мы контролируем спред брокера, проверяем наличие ограничений на торговые операции по символу, установленные брокером, и проверяем достаточно ли баров в истории для расчета советника.

```
class Trade
  {
private:
double StopLoss;
double Profit;
double   Lot;

CPositionInfo  m_position;                   // trade position object
CTrade         m_trade;                      // trading object

public:
                     Trade(double stopLoss, double profit, double lot);
                    ~Trade();
void                 Order(bool Buy, bool StopBuy, bool Sell, bool StopSell);
  };
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
Trade::Trade(double stopLoss, double profit, double lot)
  {
  StopLoss=stopLoss;
  Profit=profit;
  Lot=lot;
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
Trade::~Trade()
  {
  }
//+------------------------------------------------------------------+
```

Код открытия и закрытия позиции выделим в класс Trade.

Здесь мы объявляем переменные экземпляра класса – стоплосс, тейкпрофит и объем торговли в лотах.

Также мы объявляем объекты библиотечных классов CPositionInfo и CTrade, которые мы будем использовать для закрытия позиций.

И здесь мы объявляем метод Order открытия позиции.

```
Trade::Order(bool Buy, bool BuyStop, bool Sell, bool SellStop){
//Проверка наличия открытой позиции, чтобы не пытаться открыть ее заново
   bool BuyOpened=false;
   bool SellOpened=false;

   if(PositionSelect(_Symbol)==true)
     {
      if(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE_BUY)
        {
         BuyOpened=true;
        }
      else if(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE_SELL)
        {
         SellOpened=true;
        }
     }
     //--------------------------------------------------------------------------
MqlTradeRequest mrequest;
MqlTradeCheckResult check_result;
MqlTradeResult mresult;

MqlTick latest_price;
if(!SymbolInfoTick(_Symbol,latest_price))
     {
      Alert("Ошибка получения последних котировок - ошибка:",GetLastError(),"!!");
      return;
     }
if(Buy==true&&BuyOpened==false){
```

В методе Order мы используем входные параметры метода как флаги на открытие или закрытие позиции на покупку или продажу.

```
class Sidus
  {
private:
int numberBarOpenPosition;
int numberBarStopPosition;
int     handleIMA18;
double     MA18Buffer[];
int     handleIMA28;
double     MA28Buffer[];
int     handleIWMA5;
double     WMA5Buffer[];
int     handleIWMA8;
double     WMA8Buffer[];

public:
                     Sidus(int BarOpenPosition, int BarStopPosition);
                    ~Sidus();
bool OnTradeSignalBuy();
bool OnTradeSignalBuyStop(MqlRates& mrate[]);
bool OnTradeSignalSell();
bool OnTradeSignalSellStop(MqlRates& mrate[]);
  };
//+------------------------------------------------------------------+
Sidus::Sidus(int BarOpenPosition, int BarStopPosition)
  {
  numberBarOpenPosition=BarOpenPosition;
  numberBarStopPosition=BarStopPosition;
  handleIMA18=iMA(_Symbol,PERIOD_H1,18,0,MODE_EMA,PRICE_WEIGHTED);
  handleIMA28=iMA(_Symbol,PERIOD_H1,28,0,MODE_EMA,PRICE_WEIGHTED);
  handleIWMA5=iMA(_Symbol,PERIOD_H1,5,0,MODE_LWMA,PRICE_WEIGHTED);
  handleIWMA8=iMA(_Symbol,PERIOD_H1,8,0,MODE_LWMA,PRICE_WEIGHTED);
  }
```

И функции сигналов торговой системы мы выделим в класс Sidus.

Здесь мы объявляем переменные экземпляра класса – хэндлы используемых индикаторов и их буферы.

И в конструкторе класса мы их инициализируем.

И мы также объявляем и реализуем методы вычисления сигналов на открытие или закрытие позиции на покупку или продажу.

```mql5
#include "CheckTrade.mqh"
#include "Trade.mqh"
#include "Sidus.mqh"

input double   Lot=1;
input int      EA_Magic=1000;
input double spreadLevel=10.0;
input double StopLoss=0.01;
input double Profit=0.01;
input int numberBarOpenPosition=5;
input int numberBarStopPosition=5;

CheckTrade checkTrade;
Trade trade(StopLoss,Profit,Lot);
Sidus sidus(numberBarOpenPosition,numberBarStopPosition);

bool flagStopLoss=false;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
return(checkTrade.OnCheckTradeInit(Lot));
  }
```

В результате выделения функций в отдельные классы код советника существенно сократится.

Здесь мы сначала включаем файлы созданных нами классов с помощью директивы include.

Затем мы объявляем входные параметры советника и создаем экземпляры классов, передавая в конструкторы классов соответствующие входные параметры.

В методе OnInit советника мы просто теперь вызываем метод OnCheckTradeInit класса CheckTrade.

```mql5
void OnTick()
  {
if(!checkTrade.OnCheckTradeTick(Lot,spreadLevel)){
return;
}
//Ограничить вычисления советника по появлению нового бара на графике
static datetime last_time;
datetime last_bar_time=(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_LASTBAR_DATE);
if(last_time!=last_bar_time)
{
last_time=last_bar_time;
}else{
return;
}
//Ограничить вычисления советника по flagStopLoss
static datetime last_time_daily;
datetime last_bar_time_daily=(datetime)SeriesInfoInteger(Symbol(),PERIOD_D1,SERIES_LASTBAR_DATE);
if(last_time_daily!=last_bar_time_daily)
{
last_time_daily=last_bar_time_daily;
flagStopLoss=false;
}

if(flagStopLoss==true)return;
```

В методе OnTick советика для проверок мы вызываем метод OnCheckTradeTick класса CheckTrade.

```
//Для вычисления сигналов торговой системы требуются исторические данные символа
int num=5;
if(numberBarOpenPosition>numberBarStopPosition)num=numberBarOpenPosition;
if(numberBarOpenPosition<=numberBarStopPosition)num=numberBarStopPosition;
MqlRates mrate[];
ResetLastError();
if(CopyRates(Symbol(),Period(),0,num,mrate)<0)
     {
Print(GetLastError());
      return;
     }

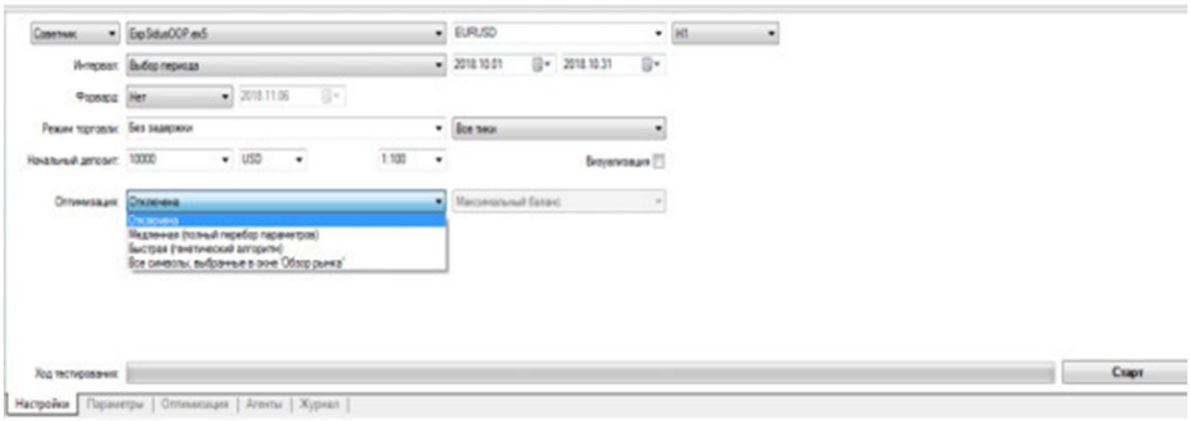
ArraySetAsSeries(mrate,true);
//---------------------------------------------------------------------------
bool TradeSignalBuy=false;
bool TradeSignalSell=false;
TradeSignalBuy=sidus.OnTradeSignalBuy();
TradeSignalSell=sidus.OnTradeSignalSell();
bool TradeSignalBuyStop=false;
bool TradeSignalSellStop=false;
TradeSignalBuyStop=sidus.OnTradeSignalBuyStop(mrate);
TradeSignalSellStop=sidus.OnTradeSignalSellStop(mrate);
trade.Order(TradeSignalBuy,TradeSignalBuyStop,TradeSignalSell,TradeSignalSellStop);
}
```

И после получения исторических данных символа, мы вычисляем сигналы торговой системы, используя методы класса Sidus.

И отправляем запрос на ордер с помощью класса Trade.

Тестирование советников

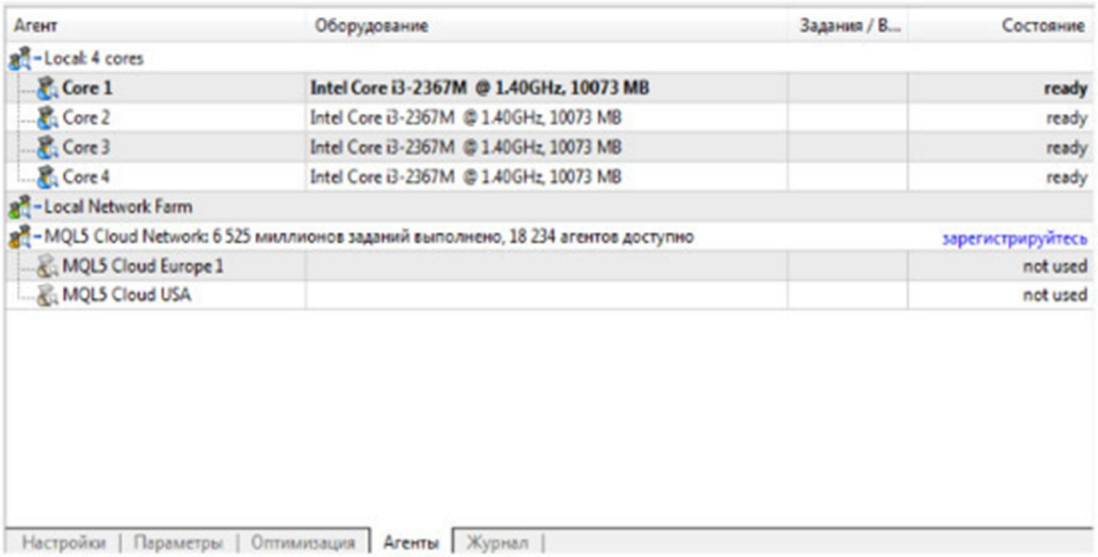
Встроенный тестер терминала MetaTrader 5 позволяет протестировать и оптимизировать входные input параметры советника с использованием исторических данных финансовых инструментов.



Открыть тестер можно, нажав правой кнопкой мышки на советнике в окне Навигатор терминала и в контекстном меню выбрав Тестировать.

Также тестер можно открыть в меню Вид терминала с помощью команды Тестер стратегий.

Тестирование является многопоточным и проводится с помощью специальных сервисов-агентов, которые представлены тремя типами.



Это локальные агенты, устанавливаемые вместе с клиентским терминалом.

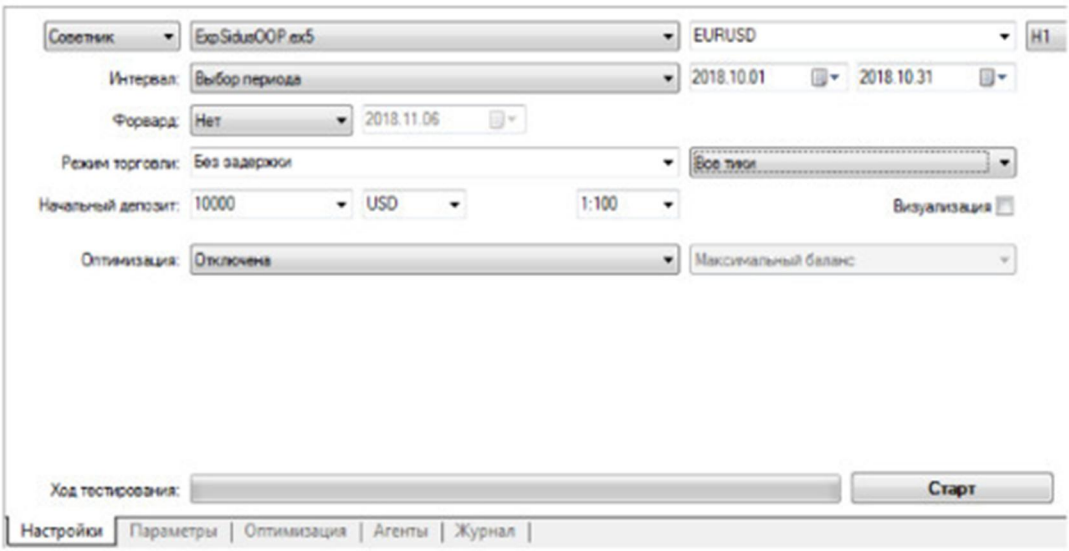
И число локальных агентов равно числу логических ядер процессора компьютера, которые используются агентами параллельно, поэтому при тестировании задействуются все доступные ресурсы компьютера.

Также для тестирования можно использовать локальные сетевые агенты, установленные на других компьютерах вместе с клиентскими терминалами.

Компьютеры должны быть соединены в локальную сеть, после чего создается ферма агентов с помощью менеджера агентов в меню Сервис терминала и производится подключение агентов во вкладе Агенты тестера терминала.

Локальные сетевые агенты также могут быть установлены на компьютере отдельно от торговой платформы с помощью файла metatester64.exe. Локальные сетевые агенты можно устанавливать только в 64-битных системах.

Еще один тип агентов это облачные агенты, использование которых платное и подключить которые также можно во вкладе Агенты тестера терминала.



Окно тестера позволяет выбрать интервал тестирования, финансовый инструмент для тестирования, если он явно не задан в советнике, временной период финансового инструмента, включить оптимизацию входных параметров советника, визуальный режим тестирования и др.

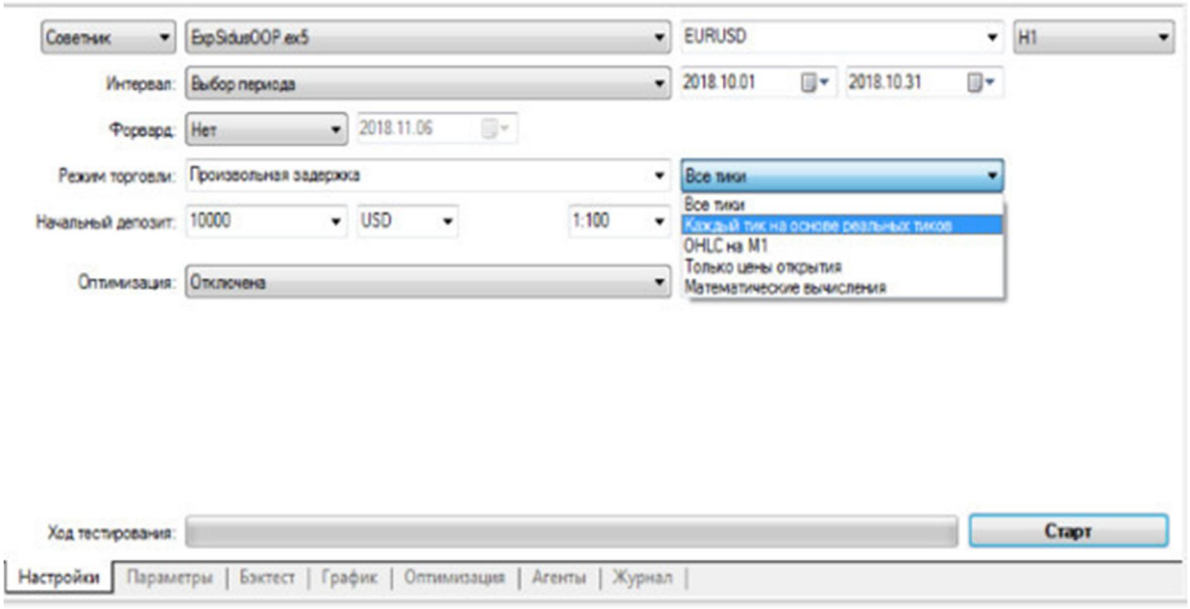


При включении визуального режима тестирования с помощью флажка Визуализация, на графике показываются используемые советником индикаторы.

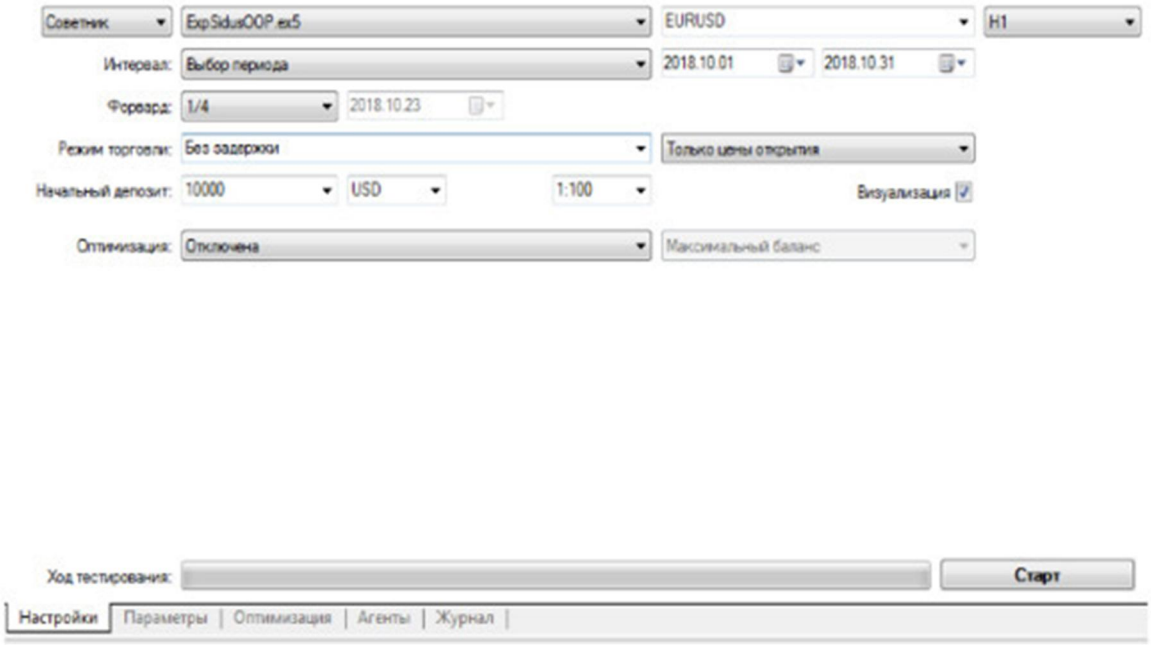
Качество истории	100%				
Бары	528	Тики	2007581	Символы	1
Начальный депозит	10 000.00				
Чистая прибыль	-406.00	Абсолютная просадка по б...	406.00	Абсолютная просадка по с...	507.00
Общая прибыль	794.28	Максимальная просадка п...	812.67 (7.81%)	Максимальная просадка п...	1 157.67 (10.87%)
Общий убыток	-1 200.28	Относительная просадка п...	7.81% (812.67)	Относительная просадка п...	10.87% (1 157.67)
Прибыльность	0.66	Матожидание выигрыша	-29.00	Уровень маржи	827.03%
Фактор восстановления	-0.35	Коэффициент Шарпа	-0.15	Z-Счет	2.48 (98.69%)
AHPR	0.9972 (-0.28%)	LR Correlation	-0.74	Результат OnTester	0
GHPR	0.9970 (-0.30%)	LR Standard Error	152.57		
Всего трейдов	14	Короткие трейды (% выигр...	6 (50.00%)	Длинные трейды (% выигр...	8 (25.00%)
Всего сделок	28	Прибыльные трейды (% от...	5 (35.71%)	Убыточные трейды (% от в...	9 (64.29%)
	Самый большой	прибыльный трейд	520.67	убыточный трейд	-247.00
	Средний	прибыльный трейд	158.86	убыточный трейд	-133.36
	Максимальное к...	непрерывных выигрышей ...	1 (520.67)	непрерывных проигрыше...	3 (-407.00)
	Макс.	непрерывная прибыль (чи...	520.67 (1)	непрерывный убыток (чис...	-407.00 (3)

Настройки | Параметры | Бэктест | График | Оптимизация | Агенты | Журнал

Вкладка Бэктест окна тестера показывает результаты тестирования советника.



Для максимальной приближенности к условиям реальной торговли, при тестировании можно выбрать режим «Произвольная задержка», «Каждый тик на основе реальных тиков».



Быстро провести оптимизацию входных параметров можно в режиме «Форвард: 1/4», «Без задержки», «Только цены открытия».

«Только цены открытия» – расчет ведется только по ценам открытия баров.

Форвард-тестированием называется повторный прогон советника на другом временном периоде.

Такая возможность предусмотрена для исключения подгонки параметров советников на определенных участках исторических данных.

С включением этой опции история котировок валют и акций делится на две части.

Непосредственно оптимизация происходит на первом отрезке истории, а второй используется только для подтверждения полученных результатов.

Если на обоих отрезках эффективность торгового робота одинаково высока, значит, торговая система обладает наилучшими параметрами и подгонка параметров практически исключена.

Форвард 1/4 – используется четверть указанного периода для форвард-тестирования.

Режим произвольных задержек исполнения эмулирует сетевые задержки при передаче и обработке торговых запросов, а также моделирует задержки исполнения приказов дилерами при реальной торговле.

Важной функцией Тестера стратегий является оптимизация торгового робота, которая позволяет подобрать для конкретного советника лучшие входные параметры.

В процессе оптимизации происходит тестирование одного торгового робота с разными входными параметрами.

Количество комбинаций входных параметров при оптимизации может достигать десятков или сотен тысяч.

В итоге, оптимизация может превратиться в очень длительный процесс, который все же можно существенно сократить при помощи генетических алгоритмов.

Эта функция отключает последовательный перебор всех комбинаций входных параметров и выбирает только те, которые наилучшим образом отвечают критериям оптимизации.

На последующих этапах «оптимальные» комбинации скрещиваются до тех пор, пока результаты не перестанут улучшаться.

Таким образом, количество комбинаций и общее время оптимизации сокращаются в разы.



Для того чтобы оценить эффективность советника создадим индикатор, показывающий все потенциальные сделки на покупку и продажу инструмента.

Данный индикатор будет основываться на скользящей средней.

При возрастании значения скользящей средней будет рассчитываться позиция на покупку, при убывании значения скользящей средней будет идти расчет позиции на продажу.

```
if((InBuffer[i]-InBuffer[i-shift_delta])>delta){
ColorBuffer[i]=2;
if(flagBuy==false){
priceBuyOpen=open[i];
 if(!ObjectCreate(0,"Buy"+i,OBJ_ARROW,0,time[i],high[i]))
      {
       return(false);
      }

       ObjectSetInteger(0,"Buy"+i,OBJPROP_COLOR,clrGreen);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_ARROWCODE,233);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_WIDTH,1);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_ANCHOR,ANCHOR_UPPER);
       ObjectSetInteger(0,"Buy"+i,OBJPROP_HIDDEN,true);
       ObjectSetString(0,"Buy"+i,OBJPROP_TOOLTIP,""+close[i]);
```

```
if((InBuffer[i-shift_delta]-InBuffer[i])>delta){
ColorBuffer[i]=1;
if(flagSell==false){
priceSellOpen=open[i];
if(!ObjectCreate(0,"Sell"+i,OBJ_ARROW,0,time[i],low[i]))
      {
       return(false);
      }
       ObjectSetInteger(0,"Sell"+i,OBJPROP_COLOR,clrRed);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_ARROWCODE,234);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_WIDTH,1);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_ANCHOR,ANCHOR_LOWER);
       ObjectSetInteger(0,"Sell"+i,OBJPROP_HIDDEN,true);
       ObjectSetString(0,"Sell"+i,OBJPROP_TOOLTIP,""+close[i]);
```

Здесь если цена начинает расти, мы встаем в покупку, если цена падает, мы встаем в продажу, и считаем потенциальный профит, который мы бы в идеале смогли бы заработать.

```
int handleProfit;
double ProfitBuffer[];
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
handleProfit=iCustom(_Symbol,PERIOD_H1,"Profit",5,0.0001,1);
return(checkTrade.OnCheckTradeInit(Lot));
  }
```

Присоединим данный индикатор к советнику Сидус, рассмотренному в предыдущем примере.

```
//+------------------------------------------------------------------+
void OnTrade()
  {
static int _deals;
ulong _ticket=0;

if(HistorySelect(0,TimeCurrent()))
  {
 int  i=HistoryDealsTotal()-1;

   if(_deals!=i) {
   _deals=i;
   } else { return; }

   if((_ticket=HistoryDealGetTicket(i))>0)
     {
      string _comment=HistoryDealGetString(_ticket,DEAL_COMMENT);
      if(StringFind(_comment,"sl",0)!=-1) {
      flagStopLoss=true;
      }
        if(HistoryDealGetDouble(_ticket, DEAL_PROFIT)!=0.0){
      if(!ObjectCreate(0,"Deal"+_ticket,OBJ_TEXT,0,
      HistoryDealGetInteger(_ticket,DEAL_TIME),HistoryDealGetDouble(_ticket,DEAL_PRICE)))
     {
      return;
     }
      ObjectSetString(0,"Deal"+_ticket,OBJPROP_TEXT,"Profit: "+HistoryDealGetDouble(_ticket, DEAL_PROFIT));
      ObjectSetDouble(0,"Deal"+_ticket,OBJPROP_ANGLE,90.0);
      ObjectSetInteger(0,"Deal"+_ticket,OBJPROP_COLOR,clrYellow);
      }}}}
```

В функции OnTrade советника будем показывать фактически полученный профит от сделки.

Произведем визуальное тестирование советника.

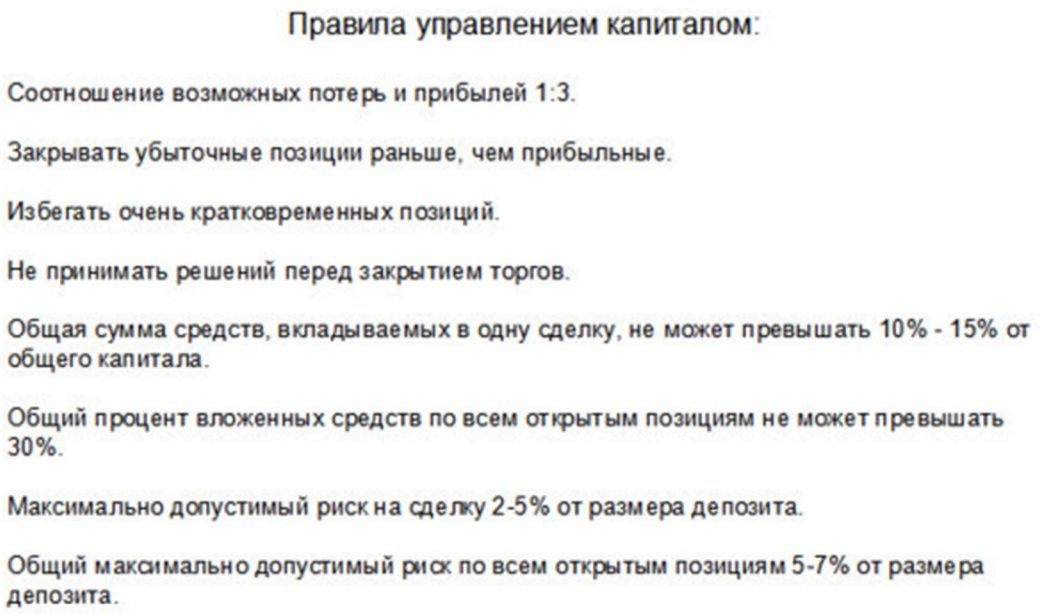


Из визуального тестирования видно, что у данного советника есть проблемы с входом и выходом из рынка, а также много пропущенных потенциальных сделок.

Управление капиталом и оценка эффективности эксперта

Управление капиталом – это способ принятия решения о том, какой частью счета следует рисковать в отдельной торговой возможности.

Типичные правила управлением капитала:



– Соотношение возможных потерь и прибылей 1:3.

– Закрывать убыточные позиции раньше, чем прибыльные.

– Избегать очень кратковременных позиций.

– Не принимать решений перед закрытием торгов.

– Общая сумма средств, вкладываемых в одну сделку, не может превышать 10% – 15% от общего капитала.

– Общий процент вложенных средств по всем открытым позициям не может превышать 30%.

– Максимально допустимый риск на сделку 2—5% от размера депозита.

– Общий максимально допустимый риск по всем открытым позициям 5—7% от размера депозита.

И типичные стратегии управления капиталом:



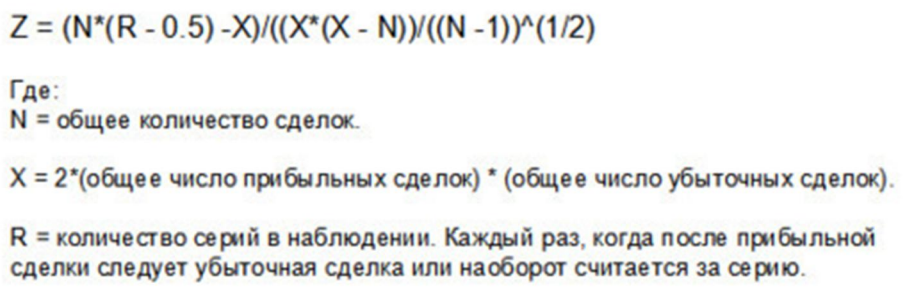
– Фиксированная сумма или фиксированный процент капитала, подвергаемый риску при следующей сделке.

– Согласование выигрышей и проигрышей при торговле. По этой методике трейдеры определяют объем торговли после успешных выигрышей или проигрышей. Например, после проигранной сделки, они могут решить удвоить объем торговли после следующего сигнала к торговле, чтобы возместить убытки. В этой системе используется статистический анализ для того, чтобы установить взаимосвязь между проигрышем и выигрышем, когда проигрыши и выигрыши или чередуются, или после определенного количества проигрышей следуют выигрыши. При этом можно после выигрыша увеличивать размер открываемых позиций и уменьшать после проигрыша, или после череды проигрышей увеличивать объем позиции, ожидая скорого выигрыша, либо, наоборот, уменьшать, ожидая проигрыша.

– Пересечение кривых цены – по этой системе производится анализ торговой системы с помощью длинной и короткой скользящих средних кривых (например, 3 и 8) прибылей и убытков сделок. Если короткая скользящая средняя кривая находится над более длинной скользящей кривой, тогда торговая система дает лучшие результаты, если же короткая скользящая средняя кривая находится под длинной скользящей кривой, тогда система работает хуже и сигналы торговой системы на открытие позиций нужно пропускать. При этом для расчета кривых используются все сигналы торговой системы, а не только те, которые были фактически реализованы. Наличие выигрышных и проигрышных периодов торговой системы также определяется с помощью статистического анализа.

Optimal f и Secure f – это определение на основе тестирования торговой системы на прошлых сделках некоторой оптимальной доли начального капитала, инвестируемой в каждую сделку, с целью достижения максимальной доходности данной системы в качестве основного критерия (Optimal f) или достижения максимальной доходности с учётом ограничения предельно допустимых убытков (Secure f).

Зависимость между проигрышами и выигрышами анализируется с помощью Z-счета. И доверительный интервал при этом должен превышать 94%.



Z-счет рассчитывается путем сравнения количества серий в наборе сделок относительно количества серий, которое можно было бы ожидать при статистической независимости результатов текущей сделки от прошедших сделок.

Z = (N* (R – 0.5) -X) / ((X* (X – N)) / ((N -1)) ^ (1/2)

Где:

N = общее количество сделок.

X = 2* (общее число прибыльных сделок) * (общее число убыточных сделок).

R = количество серий в наблюдении. Каждый раз, когда после прибыльной сделки следует убыточная сделка или наоборот считается за серию.

Положительный Z-счет означает, что в наблюдаемой истории сделок количество серий меньше, чем следовало бы ожидать при статистически независимом распределении исходов сделок.

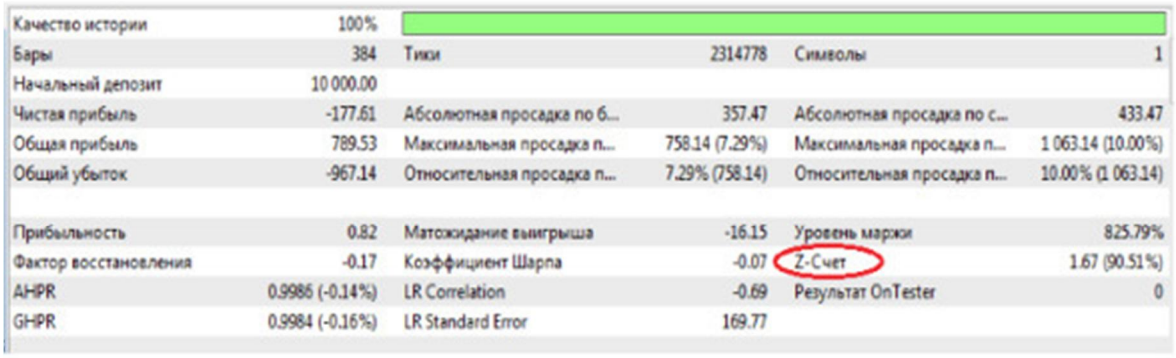
Следовательно, есть тенденция к тому, чтобы после выигрыша следовал проигрыш, а после проигрыша – выигрыш.

Наличие положительного Z-счета означает, что после убыточной сделки можно по следующему сигналу увеличить размер открываемой позиции, это позволит покрыть убытки и увеличить общую прибыльность системы.

Отрицательный Z-счет означает, что в наблюдаемой истории сделок количество серий больше, чем следовало бы ожидать при статистически независимом распределении исходов сделок.

Следовательно, за выигрышной сделкой обычно следует другая выигрышная сделка, а за убыточной сделкой следует другая убыточная сделка.

Наличие отрицательного Z-счета означает, что можно использовать пересечение кривых доходности, при том торговая система будет работать во время прибыльной фазы и выключаться после наступления убыточной фазы.



Качество истории	100%				
Бары	384	Тики	2314778	Символы	1
Начальный депозит	10 000.00				
Чистая прибыль	-177.61	Абсолютная просадка по б...	357.47	Абсолютная просадка по с...	433.47
Общая прибыль	789.53	Максимальная просадка п...	758.14 (7.29%)	Максимальная просадка п...	1 063.14 (10.00%)
Общий убыток	-967.14	Относительная просадка п...	7.29% (758.14)	Относительная просадка п...	10.00% (1 063.14)
Прибыльность	0.82	Матожидание выигрыша	-16.15	Уровень маржи	825.79%
Фактор восстановления	-0.17	Коэффициент Шарпа	-0.07	Z-Счет	1.67 (90.51%)
AHPR	0.9986 (-0.14%)	LR Correlation	-0.69	Результат OnTester	0
GHPR	0.9984 (-0.16%)	LR Standard Error	169.77		

Значение Z-счета эксперта можно посмотреть во вкладке Бэктест тестера клиентского терминала после тестирования эксперта.

В рассмотренном примере эксперта на основе торговой системы Сидуса, при значениях numberBarOpenPosition=5 и numberBarStopPosition=5, Z-счет эксперта равен 1.67 (90.51%).

Однако на этом основании скорректировать работу эксперта достаточно проблематично, так как после выигрыша может следовать не единичный проигрыш, а серия убытков и наоборот.

Вывести последовательность выигрышей и проигрышей эксперта можно в его функции OnDeinit.

```
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason){
  if(HistorySelect(0,TimeCurrent()))
  {
 int  n=HistoryDealsTotal();
 for(int i=0;i<n;i++){
  if(HistoryDealGetDouble(HistoryDealGetTicket(i), DEAL_PROFIT)!=0.0){
      Print("Profit ",HistoryDealGetDouble(HistoryDealGetTicket(i), DEAL_PROFIT));
      }
  }
 }
}
```

Общая эффективность советника определяется как разница между реализованной в ходе трейда прибылью и потенциально возможной прибылью за время этого же трейда.

Общая эффективность =
(Реализованная разница цен)/(Потенциальная прибыль)

Для длинных позиций:
Общая эффективность = (Цена закрытия - Цена открытия)/
(Максимальная цена - Минимальная цена)

Для коротких позиций:
Общая эффективность = (Цена открытия - Цена закрытия)/
(Максимальная цена - Минимальная цена)

Общая эффективность =

(Реализованная разница цен) / (Потенциальная прибыль)

Для длинных позиций:

Общая эффективность = (Цена закрытия – Цена открытия) /

(Максимальная цена – Минимальная цена)

Для коротких позиций:

Общая эффективность = (Цена открытия – Цена закрытия) /

(Максимальная цена – Минимальная цена)

Эффективность входа в рынок определяется как максимальная разница в ценах относительно цены входа как часть общего потенциала доходности в ходе трейда.

Эффективность входа = (максимальная разница в ценах относительно цены входа)/(Потенциальная прибыль)

Для длинных позиций:
Эффективность входа =
(Максимальная цена - Цена открытия трейда)/(Максимальная цена - Минимальная цена)

Для коротких позиций:
Эффективность входа =
(Цена открытия трейда - Минимальная цена)/(Максимальная цена - Минимальная цена)

Эффективность входа в рынок показывает, насколько хорошо торговая система открывает трейды – в случае длинных позиций это то, насколько сигнал входа близок к наименьшей точке в ходе трейда, в случае коротких позиций – насколько сигнал входа близок к максимальной точке в ходе трейда.

Эффективность входа = (максимальная разница в ценах относительно цены входа) / (Потенциальная прибыль)

Максимальная разница в ценах относительно цены входа – это разница между максимальной ценой и ценой входа (для коротких позиций – минимальной ценой).

Для длинных позиций:

Эффективность входа = (Максимальная цена – Цена открытия трейда) / (Максимальная цена – Минимальная цена)

Для коротких позиций:

Эффективность входа = (Цена открытия трейда – Минимальная цена) / (Максимальная цена – Минимальная цена)

Эффективность выхода из рынка определяется как максимальная разница в ценах относительно цены выхода как часть общего потенциала доходности в ходе трейда.

Эффективность выхода =
(Максимальная разница в ценах относительно цены выхода)/(Потенциальная прибыль)

Для длинных позиций:
Эффективность выхода =
(Цена выхода - Минимальная цена) / (Максимальная цена - Минимальная цена)

Для коротких позиций:
Эффективность выхода =
(Максимальная цена - Цена выхода) / (Максимальная цена - Минимальная цена)

Эффективность выхода из рынка показывает, насколько хорошо система закрывает трейды – в случае длинных позиций это то, насколько сигнал выхода близок к максимальной точке в ходе трейда, в случае коротких позиций – насколько сигнал выхода близок к минимальной точке в ходе трейда.

Эффективность выхода = (максимальная разница в ценах относительно цены выхода) / (Потенциальная прибыль)

Максимальная разница в ценах относительно цены выхода – это разница между ценой выхода и минимальной ценой (для коротких позиций – максимальной ценой).

Для длинных позиций:

Эффективность выхода = (Цена выхода – Минимальная цена) / (Максимальная цена – Минимальная цена)

Для коротких позиций:

Эффективность выхода = (Максимальная цена – Цена выхода) / (Максимальная цена – Минимальная цена)

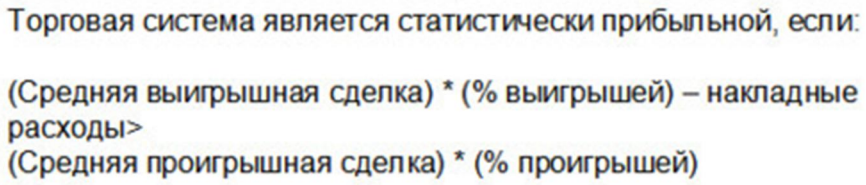
Общая эффективность является суммой эффективности входа и эффективности выхода минус 1.

Эффективность входа и эффективность выхода могут быть позитивными величинами, однако общая эффективность может быть отрицательной величиной.

Если сумма эффективности входа и эффективности выхода меньше 100%, то сделка убыточна.

Для анализа работы эксперта вычисляются средняя общая эффективность, средняя эффективность входов и средняя эффективность выходов.

Для вычисления средних величин лучше пользоваться статистическим анализом, чтобы отбрасывать выскакивающие сделки.



Торговая система является статистически прибыльной, если:

(Средняя выигрышная сделка) * (% выигрышей) – накладные расходы>

(Средняя проигрышная сделка) * (% проигрышей)

Накладные расходы это проскальзывания, комиссионные и др.

Такие показатели как Средняя выигрышная сделка (Средний прибыльный трейд), % выигрышей (Прибыльные трейды (% от всех)), Средняя проигрышная сделка (Средний убыточный трейд), % проигрышей (Убыточные трейды (% от всех)) можно посмотреть во вкладке Бэктест тестера клиентского терминала после тестирования эксперта.

Показатель статистической прибыльности эксперта без учета накладных расходов или матожидание выигрыша также можно посмотреть во вкладке Бэктест тестера клиентского терминала после тестирования эксперта.



Матожидание выигрыша рассчитывается по следующей формуле.

Expected Payoff = (ProfitTrades / TotalTrades) * (GrossProfit / ProfitTrades) —

(LossTrades / TotalTrades) * (GrossLoss / LossTrades)

где:

TotalTrades – общее количество сделок;

ProfitTrades – количество прибыльных сделок;

LossTrades – количество убыточных сделок;

GrossProfit – общая прибыль;

GrossLoss – общий убыток.

Здесь ProfitTrades / TotalTrades это % выигрышей,

GrossProfit / ProfitTrades – Средняя выигрышная сделка,

LossTrades / TotalTrades – % проигрышей,

GrossLoss / LossTrades – Средняя проигрышная сделка.

Качество истории	100%				
Бары	384	Тики	1515	Символы	1
Начальный депозит	10 000.00				
Чистая прибыль	503.02	Абсолютная просадка по б...	2 035.28	Абсолютная просадка по с...	2 172.60
Общая прибыль	9 922.76	Максимальная просадка п...	7 384.46 (44.22%)	Максимальная просадка п...	6 929.96 (40.14%)
Общий убыток	-9 419.74	Относительная просадка п...	44.22% (7 384.46)	Относительная просадка п...	40.14% (6 929.96)
Прибыльность	1.05	Матожидание выигрыша	22.86	Уровень маржи	95.70%
Фактор восстановления	0.07	Коэффициент Шарпа	0.07	Z-Счет	-3.28 (99.74%)
AHPR	1.0053 (0.53%)	LR Correlation	0.09	Результат OnTester	0
GHPR	1.0022 (0.22%)	LR Standard Error	2 535.63		
Всего трейдов	22	Короткие трейды (% выигр...	4 (100.00%)	Длинные трейды (% выигр...	18 (38.89%)
Всего сделок	44	Прибыльные трейды (% от...	11 (50.00%)	Убыточные трейды (% от в...	11 (50.00%)

В рассмотренном примере эксперта на основе торговой системы Сидуса, при значении numberBarOpenPosition=7 и numberBarStopPosition=2, матожидание выигрыша составляет 451 – 428=23.

Это меньше 10 пунктов, поэтому можно сказать, что прибыльность эксперта является неустойчивой.

В рассмотренном примере эксперта на основе торговой системы Сидуса заменим сигналы закрытия позиции на использование Trailing Stop, т.е. при достижении цены уровня безубыточности будет передвигать StopLoss.

```
16 class Trade
17   {
18 private:
19 double StopLoss;
20 double Profit;
21 double   Lot;
22 double TrailingStop;
23 double priceBuy;
24 double priceSell;
25 double slBuy;
26 double slSell;
27
28 CPositionInfo  m_position;
29 CTrade         m_trade;
30
31
32 public:
33                   Trade(double stopLoss, double profit, double lot, double trailingStop);
34                  ~Trade();
35 void              Order(bool Buy, bool StopBuy, bool Sell, bool StopSell);
36 void              Trailing();
37   };
```

Для этого изменим класс Trade.

Здесь мы объявляем дополнительную функцию Trailing.

```
if(BuyOpened==true) {
double  TBS=0;
    if(TrailingStop<SymbolInfoInteger(Symbol(),SYMBOL_TRADE_STOPS_LEVEL)*_Point){
    TBS=SymbolInfoInteger(Symbol(),SYMBOL_TRADE_STOPS_LEVEL)*_Point;
    }else{
    TBS=TrailingStop;
    }
if(
latest_price.bid-priceBuy>=TBS
){
         mrequest.action = TRADE_ACTION_SLTP;
         mrequest.price = NormalizeDouble(latest_price.ask,_Digits);
         mrequest.sl = NormalizeDouble(priceBuy,_Digits); // Stop Loss
         mrequest.tp = NormalizeDouble(latest_price.ask + Profit,_Digits);
         mrequest.symbol = _Symbol;
         mrequest.volume = Lot;
         mrequest.type_filling = ORDER_FILLING_FOK;
         mrequest.type = ORDER_TYPE_BUY;
         //--- отсылаем ордер
         if(!OrderCheck(mrequest,check_result))
```

И в этой функции при открытой позиции на покупку, если цена ушла достаточно вверх, мы передвигаем вверх и стоплосс.

Тоже самое делаем и для открытой позиции на продажу.

```

bool TradeSignalBuy=false;
bool TradeSignalSell=false;


TradeSignalBuy=sidus.OnTradeSignalBuy();
TradeSignalSell=sidus.OnTradeSignalSell();

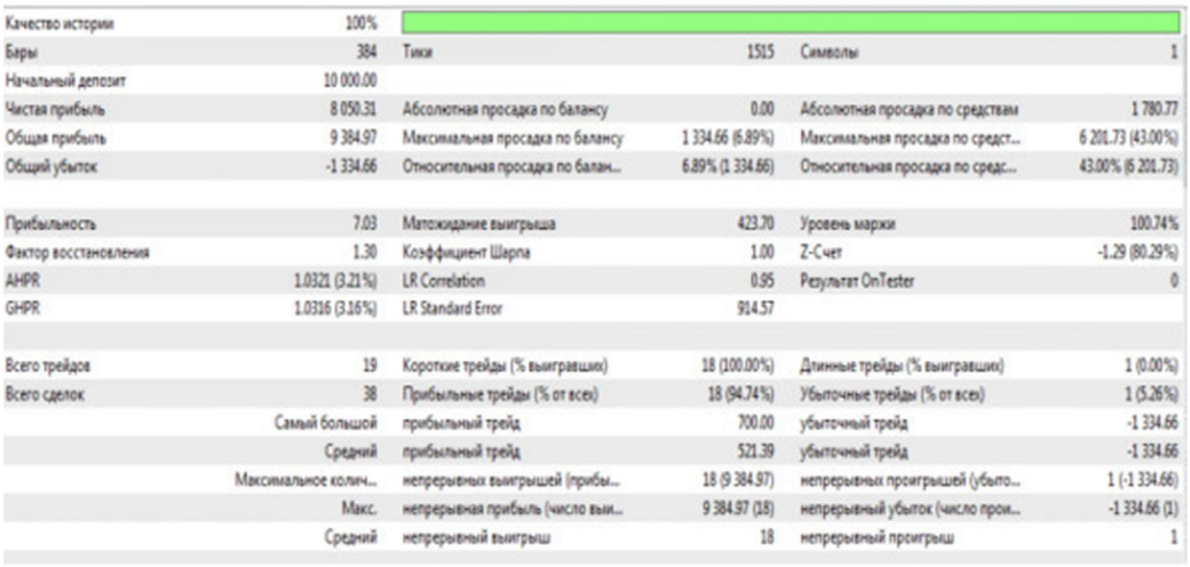
bool TradeSignalBuyStop=false;
bool TradeSignalSellStop=false;

//TradeSignalBuyStop=sidus.OnTradeSignalBuyStop();
//TradeSignalSellStop=sidus.OnTradeSignalSellStop();

trade.Order(TradeSignalBuy,TradeSignalBuyStop,TradeSignalSell,TradeSignalSellStop);
trade.Trailing();
```

В функции OnTick советника вызовем функцию Trailing класса Trade.

И протестируем советник.



Качество истории	100%				
Бары	384	Тики	1515	Символы	1
Начальный депозит	10 000.00				
Чистая прибыль	8 050.31	Абсолютная просадка по балансу	0.00	Абсолютная просадка по средствам	1 780.77
Общая прибыль	9 384.97	Максимальная просадка по балансу	1 334.66 (6.89%)	Максимальная просадка по средст...	6 201.73 (43.00%)
Общий убыток	-1 334.66	Относительная просадка по балан...	6.89% (1 334.66)	Относительная просадка по средс...	43.00% (6 201.73)
Прибыльность	7.03	Матожидание выигрыша	423.70	Уровень маржи	100.74%
Фактор восстановления	1.30	Коэффициент Шарпа	1.00	Z-Счет	-1.29 (80.29%)
AHPR	1.0321 (3.21%)	LR Correlation	0.95	Результат OnTester	0
GHPR	1.0316 (3.16%)	LR Standard Error	914.57		
Всего трейдов	19	Короткие трейды (% выигравших)	18 (100.00%)	Длинные трейды (% выигравших)	1 (0.00%)
Всего сделок	38	Прибыльные трейды (% от всех)	18 (94.74%)	Убыточные трейды (% от всех)	1 (5.26%)
	Самый большой	прибыльный трейд	700.00	убыточный трейд	-1 334.66
	Средний	прибыльный трейд	521.39	убыточный трейд	-1 334.66
	Максимальное колич...	непрерывных выигрышей (прибы...	18 (9 384.97)	непрерывных проигрышей (убыто...	1 (-1 334.66)
	Макс.	непрерывная прибыль (число вы...	9 384.97 (18)	непрерывный убыток (число прои...	-1 334.66 (1)
	Средний	непрерывный выигрыш	18	непрерывный проигрыш	1

После оптимизации стоплосса и тейкпрофита матожидание выигрыша теперь 423.

Увеличилась и чистая прибыль при трейлинге и улучшилась статистическая прибыльность эксперта.

Другой показатель вкладки Бэктест тестера стратегий, это Коэффициент Шарпа, характеризующий эффективность и стабильность эксперта.

Чем выше значение показателя, тем больше доходность на единицу риска.

Значение коэффициента Шарпа для устойчивых экспертов более 0.25.

В нашем случае коэффициент Шарпа равен 1.

Фактор роста эксперта или GHPR (среднее геометрическое сделки) будет равен (Конечный депозит/Начальный депозит) в степени 1/ (Всего трейдов).

В данном случае GHPR (фактор роста) равен 1,0316 – больше единицы, что означает возможность торговли с использованием реинвестирования.

Другие показатели эксперта, которые можно увидеть во вкладке Бэктест тестера стратегий, это:

Прибыльность (Profit Factor) – Общая прибыль/общий убыток. Рекомендуемое значение не меньше 2.

Фактор восстановления – Чистая прибыль / Максимальная просадка по средствам.

Чем больше показатель, тем менее рискованным является советник.

Многие эксперты считают, что у эффективной торговой системы фактор восстановления должен быть не менее 3.

AHPR – среднеарифметическое сделки. Положительное значение говорит о том, что торговая система прибыльна.

Уровень маржи – минимальный уровень маржи в процентах, который был зафиксирован за период тестирования.

LR Correlation – позволяет оценить отклонения точек графика баланса счета от линейной регрессии.

Чем LR Correlation ближе к нулю, тем более случайный характер имеет торговля.

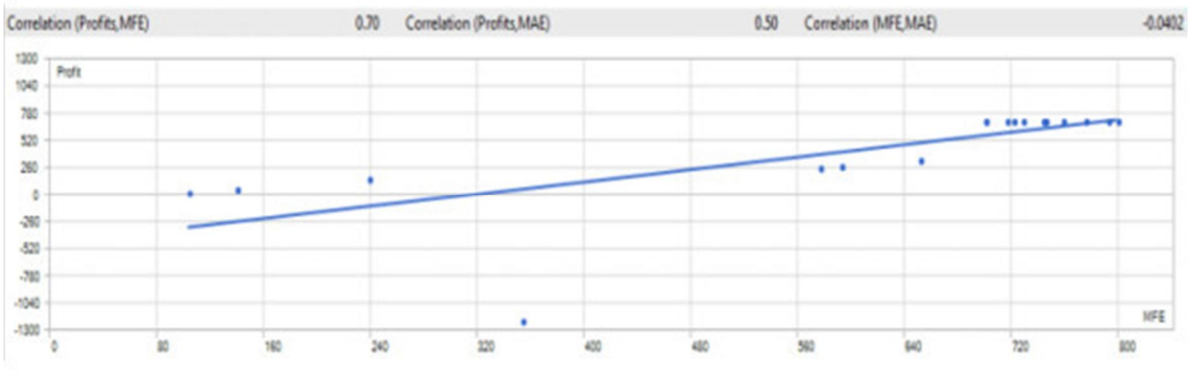
LR Standard Error – отклонение графика баланса счета от линейной регрессии в денежном выражении.

Средний прибыльный трейд / Средний убыточный трейд – желательно, чтобы отношение этих двух показателей было больше единицы.

Максимальное количество непрерывных проигрышей (убыток) – желательно, чтобы этот показатель был как можно меньше.

Исходя из глубины максимальной посадки по средствам, можно вычислить минимальный депозит, необходимый, чтобы начать торговать с данным экспертом.

Мин. депозит = Максимальная просадка по средствам * 2



Correlation (Profits, MFE) – связь между результатами позиций и MFE (Maximum Favorable Excursion – максимальный размер потенциальной прибыли, наблюдаемый за время удержания позиции).

MFE показывает максимальное движение цены в благоприятном направлении.

Чем ближе показатель Correlation (Profits, MFE) к единице, тем лучше эксперт реализует потенциальную прибыль.

Correlation (Profits, MAE) – связь между результатами позиций и MAE (Maximum Adverse Excursion – максимальный потенциальной убыток, наблюдаемый за время удержания позиции).

MAE показывает максимально неблагоприятное движение цены.

Чем ближе показатель Correlation (Profits, MAE) к единице, тем лучше эксперт использует защитный Stop Loss.

Correlation (MFE, MAE) – связь между MFE и MAE.

Чем ближе показатель Correlation (MFE, MAE) к единице, тем лучше эксперт реализует максимальную прибыль и максимально защищает позицию на всем протяжении ее жизни.

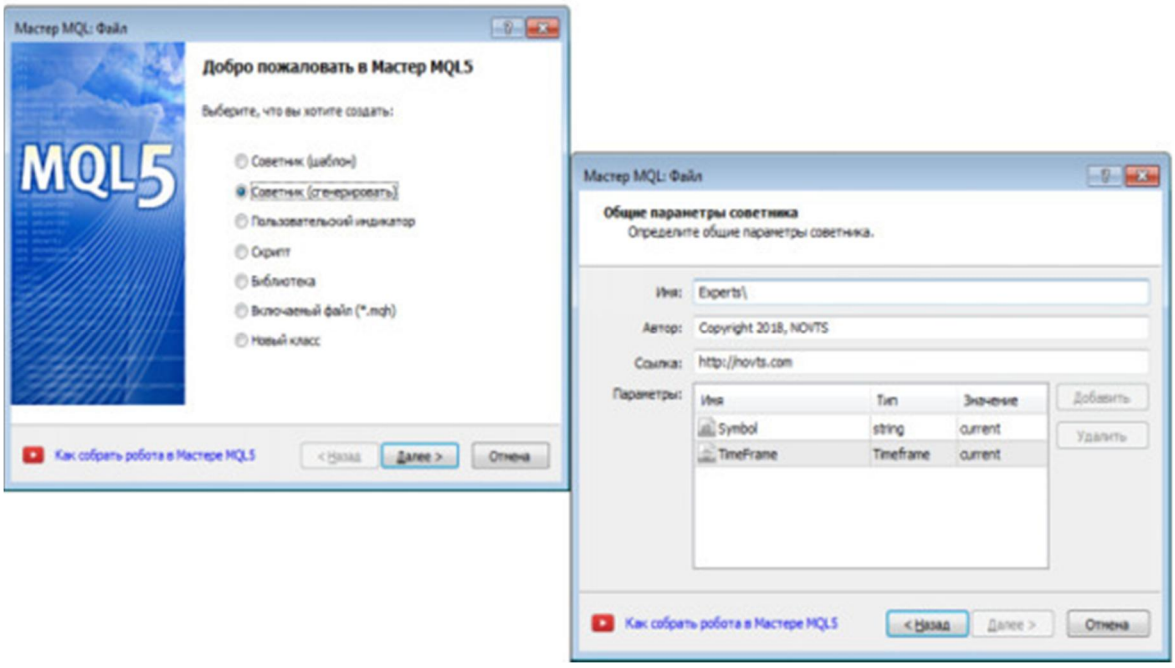


Среднее время удержания позиции – показатель рассчитывается как общее время удержания, деленное на количество сделок.

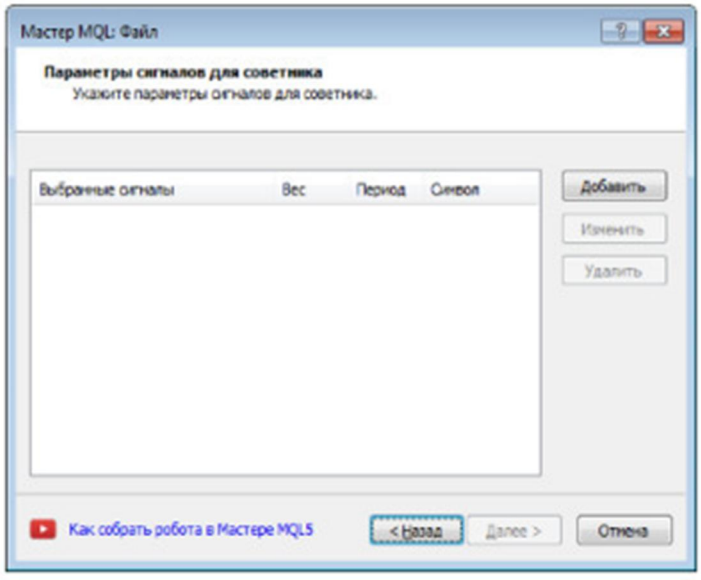
Время удержания позиции увеличивает риск, потому что просадка, очевидно, может быть большей при позициях, удерживаемых в течение большего периода времени.

Создание эксперта с помощью мастера MQL5

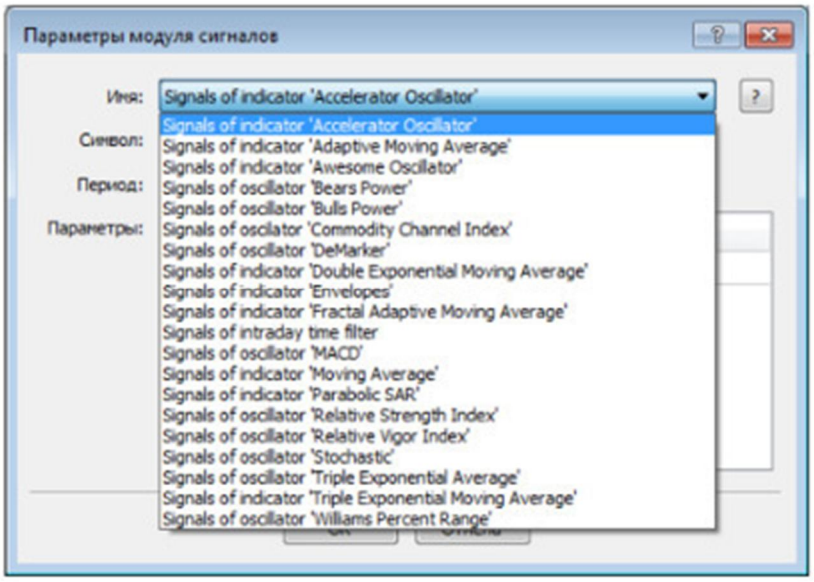
Мастер MQL5, который открывается с помощью кнопки Создать панели инструментов редактора MetaEditor, позволяет сгенерировать код эксперта на основе готовых модулей – сигналов, модулей управления капиталом и трейлинг-стопа.



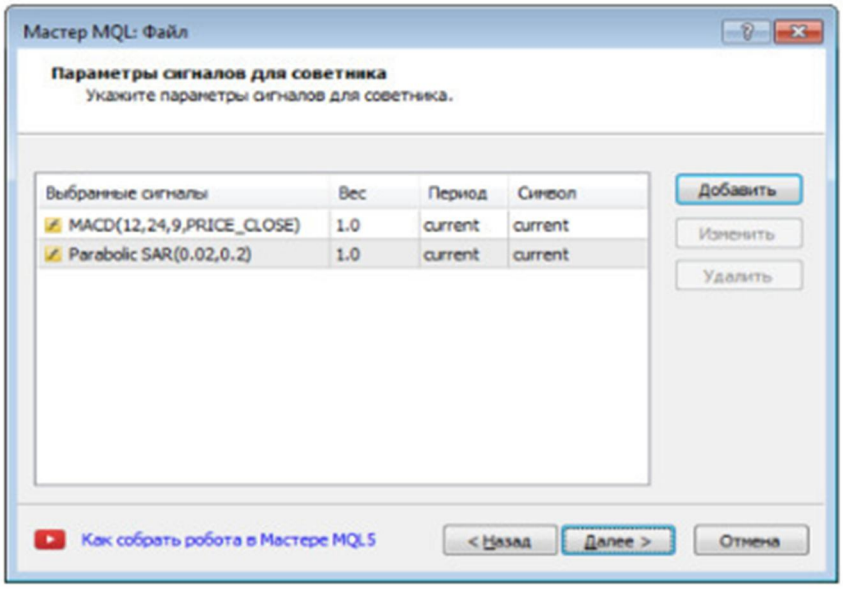
И модуль сигнала здесь добавляется с помощью кнопки Добавить.



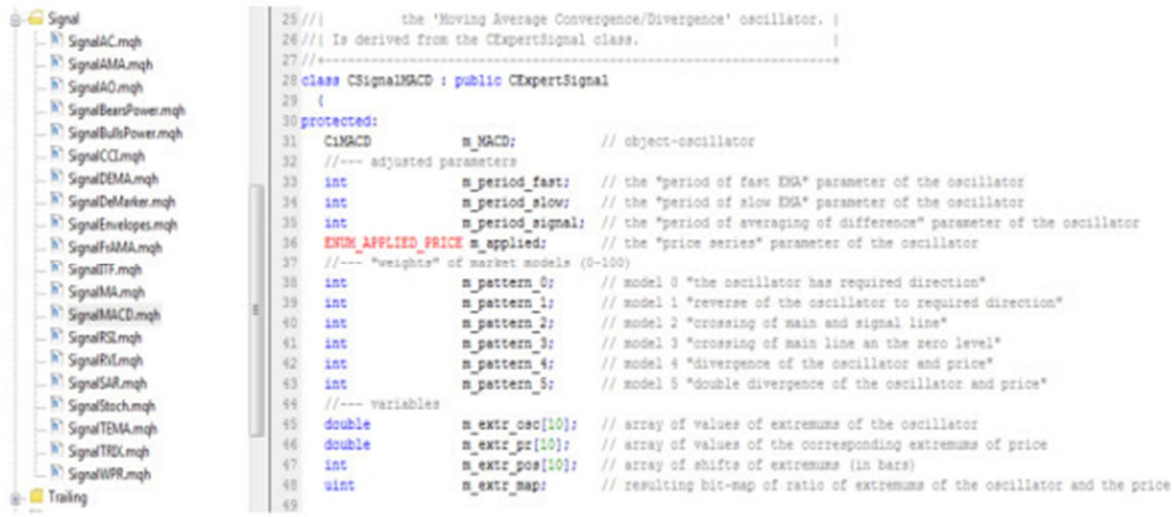
Файлы модулей сигналов – это включаемые файлы Include (*.mqh), расположенные в папке MQL5\Include\Expert\Signal.



В качестве примера выберем сигнал MACD и сигнал PSAR.



Если посмотреть файлы SignalMACD.mqh и SignalSAR.mqh папки MQL5\Include\Expert\Signal, сигнал MACD имеет 5 моделей прогноза цены.



Модель 0 – «осциллятор имеет необходимое направление» – значимость 10.

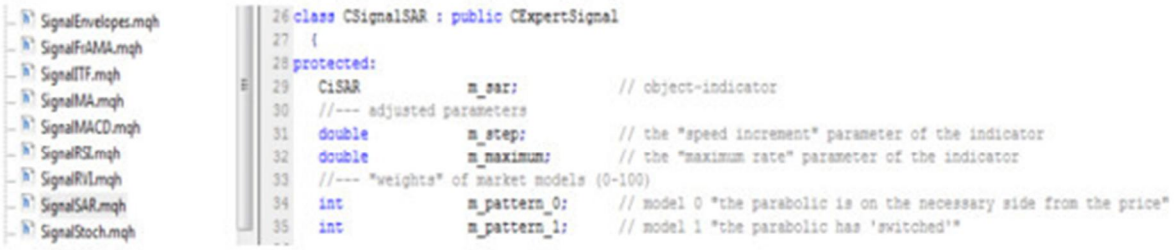
Модель 1 – «разворот осциллятора в нужном направлении» – значимость 30.

Модель 2 – «пересечение основной и сигнальной линии» – значимость 80.

Модель 3 – «пересечение главной линией нулевого уровня» – значимость 50.

Модель 4 – «дивергенция осциллятора и цены» – значимость 60.

Модель 5 – «двойная дивергенция осциллятора и цены» – значимость 100.



Сигнал SAR имеет 2 модели прогноза цены:

Модель 0 – «параболик находится на нужной стороне цены» – значимость 40.

Модель 1 – «параболик переключается на другую сторону цены» – значимость 90.

Если модель дает сигнал на падение цены – значимость отрицательная, если на рост цены – значимость положительная.

Итоговый прогноз двух модулей будет рассчитываться по следующей формуле.

Итоговый Прогноз = (Прогноз MACD + Прогноз SAR)/2

Где

Прогноз MACD = Вес сигнала MACD * значимость Модели MACD
Прогноз SAR = Вес сигнала SAR * значимость Модели SAR

Итоговый Прогноз = (Прогноз MACD + Прогноз SAR) /2

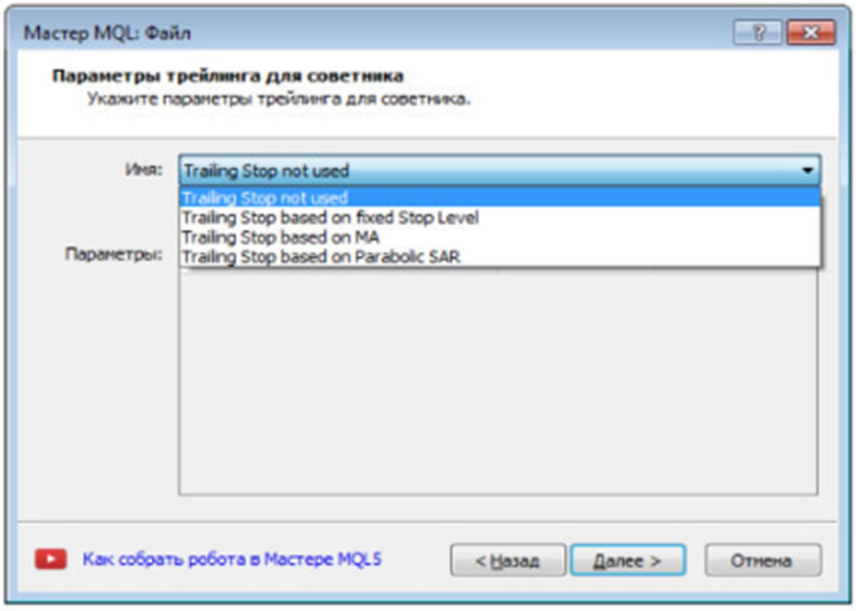
Где Прогноз MACD = Вес сигнала MACD * значимость Модели MACD,

Прогноз SAR = Вес сигнала SAR * значимость Модели SAR

В нашем случае мы установили весы сигналов равными единице.

Если итоговый прогноз превысит пороговое значение, эксперт совершит сделку на покупку или продажу.

После определения сигналов эксперта определяется алгоритм сопровождения открытой позиции.



Это

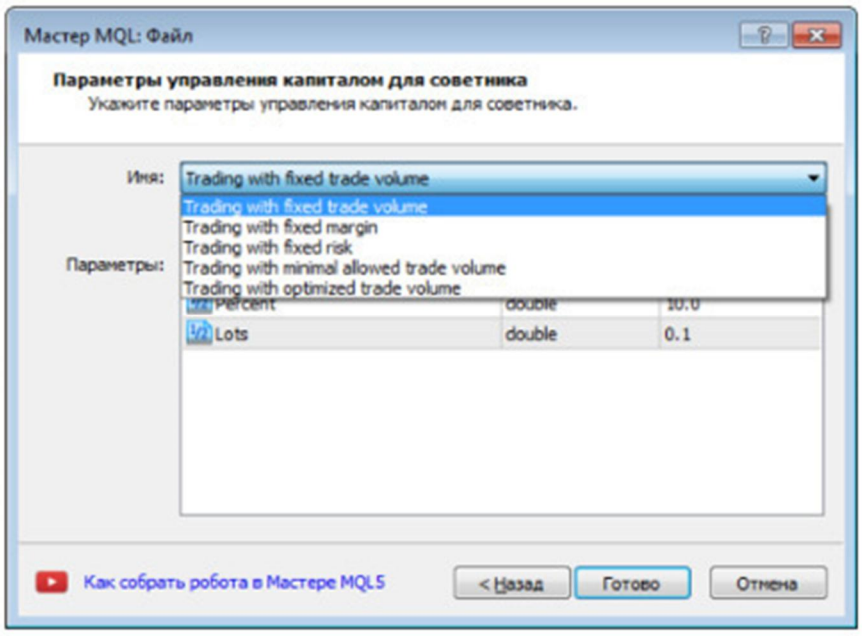
– Сопровождение открытой позиции на фиксированном «расстоянии» (в пунктах) – уровни Stop Loss и Take Profit открытой позиции перемещаются на фиксированное расстояние по движению цены в направлении открытой позиции.

Когда цена перемещается в направлении открытой позиции на расстояние, которое превышает количество пунктов, соответствующих уровню Trailing Stop Level, эксперт изменяет значения уровней Stop Loss и Take Profit (если Trailing Profit Level> 0).

– Сопровождение открытой позиции по значениям скользящей средней на предыдущем баре.

– Сопровождение открытой позиции по значениям индикатора Parabolic SAR на предыдущем баре.

И файлы реализации алгоритма сопровождения открытой позиции находятся в папке MQL5\Include\Expert\Trailing.



После определения алгоритма сопровождения открытой позиции устанавливается алгоритм управления капиталом и рисками.

Это

– Торговля с фиксированным лотом.

– Торговля с фиксированным уровнем маржи.

И здесь значение лота вычисляется функцией MaxLotCheck, которая возвращает максимально возможный объем торговой операции на основе доли свободной маржи (здесь по умолчанию 10%).

– Торговля с фиксированным уровнем риска.

И здесь значение лота вычисляется как отношение доли баланса, выделенной для риска, к StopLoss.

– Торговля минимальным лотом.

– И торговля объемом, определяемым результатами предыдущих сделок.

Здесь сначала значение лота вычисляется функцией MaxLotCheck, которая возвращает максимально возможный объем торговой операции на основе доли свободной маржи (здесь по умолчанию 10%).

Затем в случае получения убытка лот уменьшается на фактор Decrease Factor (по умолчанию 3).

Если образуется убыток, равный указанному проценту от текущего капитала, производится принудительное закрытие убыточной позиции.

Файлы реализации алгоритма управления капиталом и рисками находятся в папке MQL5\Include\Expert\Money.

После выбора алгоритма управления капиталом и рисками генерируется код эксперта.

```
#include <Expert\Expert.mqh>
//--- available signals
#include <Expert\Signal\SignalMACD.mqh>
#include <Expert\Signal\SignalSAR.mqh>
//--- available trailing
#include <Expert\Trailing\TrailingFixedPips.mqh>
//--- available money management
#include <Expert\Money\MoneyFixedLot.mqh>
//+------------------------------------------------------------------+
//| Inputs                                                           |
//+------------------------------------------------------------------+
//--- inputs for expert
input string              Expert_Title                ="ExpGen";    // Document name
ulong                     Expert_MagicNumber          =31606;       //
bool                      Expert_EveryTick            =false;       //
//--- inputs for main signal
input int                 Signal_ThresholdOpen        =10;          // Signal threshold value to open [0...100]
input int                 Signal_ThresholdClose       =10;          // Signal threshold value to close [0...100]
input double              Signal_PriceLevel           =0.0;         // Price level to execute a deal
input double              Signal_StopLevel            =50.0;        // Stop Loss level (in points)
input double              Signal_TakeLevel            =50.0;        // Take Profit level (in points)
input int                 Signal_Expiration           =4;           // Expiration of pending orders (in bars)
input int                 Signal_MACD_PeriodFast      =12;          // MACD(12,24,9,PRICE_CLOSE) Period of fast EMA
input int                 Signal_MACD_PeriodSlow      =24;          // MACD(12,24,9,PRICE_CLOSE) Period of slow EMA
input int                 Signal_MACD_PeriodSignal    =9;           // MACD(12,24,9,PRICE_CLOSE) Period of averaging of difference
input ENUM_APPLIED_PRICE  Signal_MACD_Applied          =PRICE_CLOSE; // MACD(12,24,9,PRICE_CLOSE) Prices series
input double              Signal_MACD_Weight          =1.0;         // MACD(12,24,9,PRICE_CLOSE) Weight [0...1.0]
input double              Signal_SAR_Step             =0.02;        // Parabolic SAR(0.02,0.2) Speed increment
input double              Signal_SAR_Maximum          =0.2;         // Parabolic SAR(0.02,0.2) Maximum rate
input double              Signal_SAR_Weight           =1.0;         // Parabolic SAR(0.02,0.2) Weight [0...1.0]
//--- inputs for trailing
input int                 Trailing_FixedPips_StopLevel  =30;        // Stop Loss trailing level (in points)
input int                 Trailing_FixedPips_ProfitLevel=50;        // Take Profit trailing level (in points)
//--- inputs for money
input double              Money_FixLot_Percent        =10.0;        // Percent
input double              Money_FixLot_Lots           =1.0;         // Fixed volume
```

Код эксперта основан на использовании экземпляра класса CExpert, файл которого находится в папке MQL5\Include\Expert.

Пороговые значения Signal_ThresholdOpen и Signal_ThresholdClose итогового прогноза сигналов по умолчанию равны 10.

Тестирование этого эксперта с сигналами MACD и PSAR на часовом графике EUR/USD с разными алгоритмами трейлинга и маниенеджмента дает отрицательное матожидание выигрыша.

Начальный депозит	10 000.00		
Чистая прибыль	-2 646.55	Абсолютная просадка по балансу	2 649.55
Общая прибыль	5 084.06	Максимальная просадка по балансу	3 541.30 (32.51%)
Общий убыток	-7 730.61	Относительная просадка по балансу	32.51% (3 541.30)
Прибыльность	0.66	Матожидание выигрыша	-38.36
Фактор восстановления	-0.69	Коэффициент Шарпа	-0.14
AHPR	0.9959 (-0.41%)	LR Correlation	-0.94
GHPR	0.9956 (-0.44%)	LR Standard Error	353.61

Ничего не дает и оптимизация таких параметров, как Trailing_FixedPips_StopLevel, Signal_MACD_Weight и Signal_SAR_Weight.

Эксперт остается убыточным.

Попробуем создать эксперт и включить в него все стандартные сигналы мастера MQL5 Wizard.

При тестировании на часовом графике EUR/USD такой эксперт с равными весами сигналов также покажет отрицательное матожидание выигрыша.

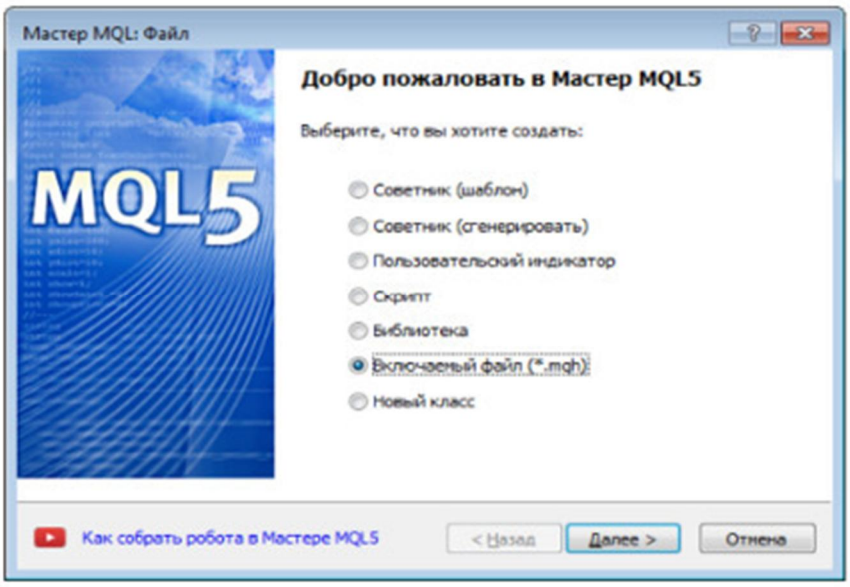
Однако при оптимизации весов сигналов, эксперт выйдет в плюс, и будут видны комбинации сигналов со значимыми весами, дающие наилучший результат.

Также можно провести оптимизацию пороговых значений прогноза вместе с оптимизацией весов сигналов.

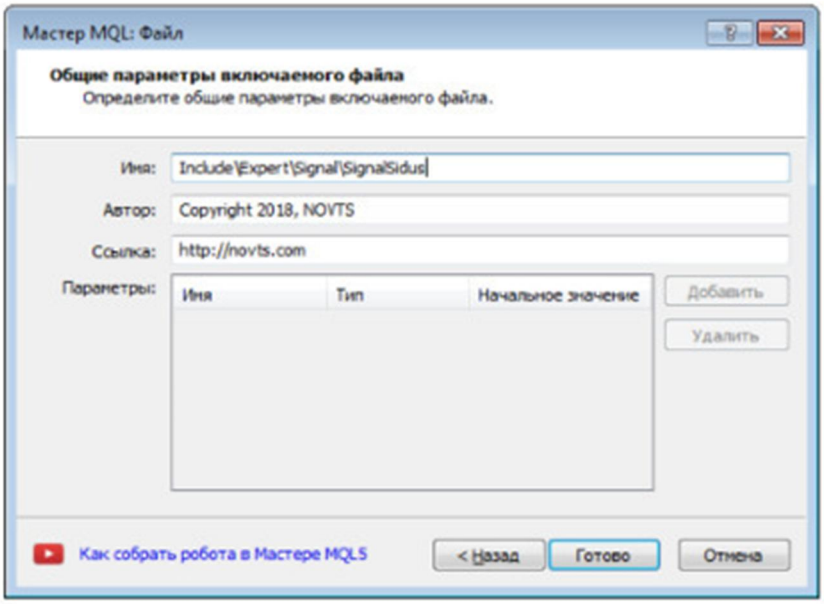
На основании полученных комбинаций сигналов, выводящих советник в прибыль, уже можно строить торговые системы.

Модульная структура эксперта, генерируемого мастером MQL5 Wizard, позволяет включить в советник свой собственный модуль сигналов торговой системы.

В качестве примера рассмотрим создание модуля сигналов на основе торговой системы Сидуса.



В редакторе MetaEditor нажмем кнопку Создать и в мастере MQL5 создадим включаемый файл.



Который поместим в папку MQL5\Include\Expert\Signal.

Включаемый файл должен содержать класс, расширяющий класс CExpertSignal.

```
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"

#include <Expert\ExpertSignal.mqh>

```

Поэтому в код необходимо включить файл ExpertSignal.mqh класса CExpertSignal, используя директиву include.

```
// wizard description start
//+------------------------------------------------------------------+
//| Description of the class                                         |
//| Title=Signals of the system 'Sidus'                              |
//| Type=SignalAdvanced                                              |
//| Name=Sidus                                                       |
//| ShortName=MA                                                     |
//| Class=CSignalSidus                                               |
//| Page=                                                            |
//| Parameter=NumberOpenPosition,int,5,Bars number checked to cross  |
//| Parameter=Pattern_0,int,80,Model 0                               |
//| Parameter=Pattern_1,int,20,Model 1                               |
//| Parameter=Pattern_2,int,80,Model 2                               |
//+------------------------------------------------------------------+
// wizard description end
//+------------------------------------------------------------------+
//| Class CSignalSidus.                                              |
//| Purpose: Class of generator of trade signals based on            |
//|          the 'Sidus' system.                                     |
//| Is derived from the CExpertSignal class.                         |
//+------------------------------------------------------------------+
```

Далее должна присутствовать информация о модуле сигналов, предназначенная для мастера MQL5, которая используется для

распознавания модуля сигналов мастером MQL5 при создании эксперта в окне добавления сигналов, а также при генерации самого кода эксперта.

Без этой информации мастер MQL5 просто не добавит сигнал в свой интерфейс, хотя файл сигнала и будет расположен в папке MQL5\Include\Expert\Signal.

После создания модуля сигналов редактор MetaEditor необходимо перезагрузить, чтобы сигнал добавился в мастер MQL5.

Здесь строка Title отображается в списке сигналов окна мастера MQL5, строка Page указывает на раздел справки и должна быть пустой, строки Parameter описывают методы установки параметров сигнала и используются при генерации кода советника.

```
class CSignalSidus : public CExpertSignal
  {
protected:
   CiMA              m_ma18;             // object-indicator
   CiMA              m_ma28;             // object-indicator
   CiMA              m_ma5;              // object-indicator
   CiMA              m_ma8;              // object-indicator
   //--- adjusted parameters
   int m_numberOpenPosition;
   //--- "weights" of market models (0-100)
   int               m_pattern_0;      // model 0 "5 WMA and 8 WMA cross the 18 EMA and the 28 EMA upward or top down" 80
   int               m_pattern_1;      // model 1 "5 WMA crosses the 8 WMA upward or top down" 10
   int               m_pattern_2;      // model 2 "18 EMA crosses the 28 EMA upward or top down" 80

public:
                     CSignalSidus(void);
                    ~CSignalSidus(void);
   //--- methods of setting adjustable parameters
   void              NumberOpenPosition(int value)               { m_numberOpenPosition=value;    }
   //--- methods of adjusting "weights" of market models
   void              Pattern_0(int value)                { m_pattern_0=value;          }
   void              Pattern_1(int value)                { m_pattern_1=value;          }
   void              Pattern_2(int value)                { m_pattern_2=value;          }
   //--- method of verification of settings
   virtual bool      ValidationSettings(void);
   //--- method of creating the indicator and timeseries
   virtual bool      InitIndicators(CIndicators *indicators);
   //--- methods of checking if the market models are formed
   virtual int       LongCondition(void);
   virtual int       ShortCondition(void);
protected:
   //--- method of initialization of the indicator
   bool              InitMA(CIndicators *indicators);
  };
```

Далее идет объявление параметров и методов класса модуля сигналов.

Здесь объекты m_ma* представляют скользящие средние, используемые системой Сидуса, параметр m_numberOpenPosition представляет количество баров, на которых будет проверяться пересечение скользящих средних.

Для генерации сигналов определим три модели прогноза цены – пересечение скользящими средними 5WMA и 8 WMA скользящих средних 18 EMA и 28 EMA, пересечение скользящей средней 5WMA скользящую среднюю 8 WMA, и пересечение скользящей средней 18 EMA скользящую среднюю 28 EMA.

И здесь метод ValidationSettings проверяет на корректность параметры сигнала, метод InitIndicators инициализирует объекты m_ma*.

Генерируемый эксперт будет получать сигналы модуля с помощью методов CheckOpenLong. CheckOpenShort, CheckCloseLong, CheckCloseShort, CheckReverseLong, CheckReverseShort класса CExpertSignal.

Однако эти методы, в свою очередь, формируют свои возвращаемые значения на основе значений, которые возвращаются методами LongCondition и ShortCondition нашего модуля.

Методы LongCondition и ShortCondition как раз и оперируют моделями прогноза цены.

Потому в модуле сигналов достаточно определить эти два метода.

```
68 CSignalSidus::CSignalSidus(void) : m_numberOpenPosition(5),
69                                    m_pattern_0(80),
70                                    m_pattern_1(10),
71                                    m_pattern_2(80)
72   {
73 //--- initialization of protected data
74    m_used_series=USE_SERIES_OPEN+USE_SERIES_HIGH+USE_SERIES_LOW+USE_SERIES_CLOSE;
75   }
76 //+------------------------------------------------------------------+
77 //| Destructor                                                       |
78 //+------------------------------------------------------------------+
79 CSignalSidus::~CSignalSidus(void)
80   {
81   }
82 //+------------------------------------------------------------------+
83 //| Validation settings protected data.                              |
84 //+------------------------------------------------------------------+
85 bool CSignalSidus::ValidationSettings(void)
86   {
87 //--- validation settings of additional filters
88    if(!CExpertSignal::ValidationSettings())
89       return(false);
90
91 //--- ok
92    return(true);
93   }
```

Далее определим конструктор класса и его методы.

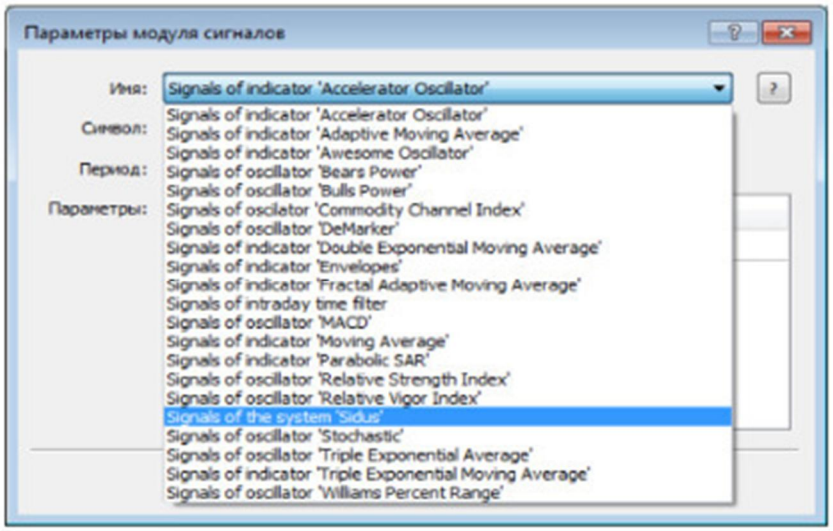
```
int CSignalSidus::LongCondition(void)
  {
   int result=0;
   int idx=StartIndex();
if(m_ma5.Main(idx)>m_ma8.Main(idx)&&m_ma8.Main(idx)>m_ma18.Main(idx)&&m_ma8.Main(idx)>m_ma28.Main(idx)){
bool flagCross1=false;
bool flagCross2=false;
for (int i=(idx+1);i<m_numberOpenPosition;i++){
if(m_ma5.Main(i)<m_ma18.Main(i)&&m_ma5.Main(i)<m_ma28.Main(i)){
 flagCross1=true;
 }
 if(m_ma8.Main(i)<m_ma18.Main(i)&&m_ma8.Main(i)<m_ma28.Main(i)){
 flagCross2=true;
 }}
 if(flagCross1==true&&flagCross2==true){
 result=m_pattern_0;
 }}
if(m_ma5.Main(idx)>m_ma8.Main(idx)){
bool flagCross=false;
for (int i=(idx+1);i<m_numberOpenPosition;i++){
if(m_ma5.Main(i)<m_ma8.Main(i)){
 flagCross=true;
 }}
 if(flagCross==true){
 result=m_pattern_1;
 }}
 if(m_ma18.Main(idx)>m_ma28.Main(idx)){
bool flagCross=false;
for (int i=(idx+1);i<m_numberOpenPosition;i++){
if(m_ma18.Main(i)<m_ma28.Main(i)){
 flagCross=true;
 }}
 if(flagCross==true){
 result=m_pattern_2;
```

Здесь в методе LongCondition мы реализуем три модели прогноза цены на покупку.

```
int CSignalSidus::ShortCondition(void)
  {
   int result=0;
   int idx   =StartIndex();
if(m_ma5.Main(idx)<m_ma8.Main(idx)&&m_ma8.Main(idx)<m_ma18.Main(idx)&&m_ma8.Main(idx)<m_ma28.Main(idx)){
bool flagCross1=false;
bool flagCross2=false;
for (int i=(idx+1);i<m_numberOpenPosition;i++){
if(m_ma5.Main(i)>m_ma18.Main(i)&&m_ma5.Main(i)>m_ma28.Main(i)){
 flagCross1=true;
 }
 if(m_ma8.Main(i)>m_ma18.Main(i)&&m_ma8.Main(i)>m_ma28.Main(i)){
 flagCross2=true;
 }}
 if(flagCross1==true&&flagCross2==true){
 result=m_pattern_0;
 }}
if(m_ma5.Main(idx)<m_ma8.Main(idx)){
bool flagCross=false;
for (int i=(idx+1);i<m_numberOpenPosition;i++){
if(m_ma5.Main(i)>m_ma8.Main(i)){
 flagCross=true;
 }}
 if(flagCross==true){
 result=m_pattern_1;
 }}
if(m_ma18.Main(idx)<m_ma28.Main(idx)){
bool flagCross=false;
for (int i=(idx+1);i<m_numberOpenPosition;i++){
if(m_ma18.Main(i)>m_ma28.Main(i)){
 flagCross=true;
 }}
 if(flagCross==true){
 result=m_pattern_2;
```

И в методе ShortCondition мы реализуем три модели прогноза цены на продажу.

С помощью мастера MQL5 сгенерируем код эксперта на основе созданного модуля сигналов.



При оптимизации параметров данного эксперта на часовом графике EUR/USD получим, что эксперт лучше всего работает со следующими значениями параметров:

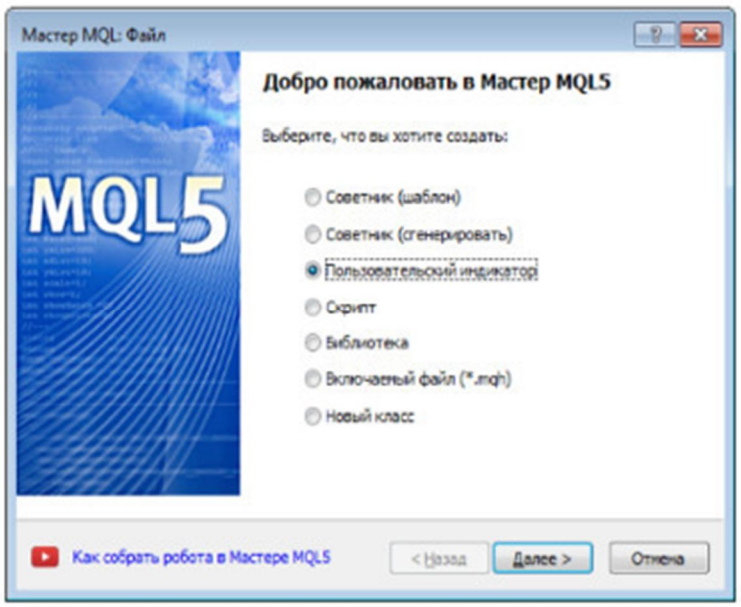
```
//--- inputs for main signal
input int    Signal_ThresholdOpen          =20;      // Signal threshold value to open [0...100]
input int    Signal_ThresholdClose         =20;      // Signal threshold value to close [0...100]
input double Signal_PriceLevel             =0.0;     // Price level to execute a deal
input double Signal_StopLevel              =50.0;    // Stop Loss level (in points)
input double Signal_TakeLevel              =50.0;    // Take Profit level (in points)
input int    Signal_Expiration             =4;       // Expiration of pending orders (in bars)
input int    Signal_MA_NumberOpenPosition  =3;       // Sidus(5,80,20,80) Bars number checked to cross
input int    Signal_MA_Pattern_0           =60;      // Sidus(5,80,20,80) Model 0
input int    Signal_MA_Pattern_1           =10;      // Sidus(5,80,20,80) Model 1
input int    Signal_MA_Pattern_2           =100;      // Sidus(5,80,20,80) Model 2
input double Signal_MA_Weight              =1.0;     // Sidus(5,80,20,80) Weight [0...1.0]
//--- inputs for trailing
input int    Trailing_FixedPips_StopLevel  =100;      // Stop Loss trailing level (in points)
input int    Trailing_FixedPips_ProfitLevel=50;      // Take Profit trailing level (in points)
```

Signal_ThresholdOpen =20;
Signal_ThresholdClose =20;
Signal_MA_NumberOpenPosition =3;

Signal_MA_Pattern_0 =60;
Signal_MA_Pattern_1 =10;
Signal_MA_Pattern_2 =100;
Trailing_FixedPips_StopLevel =100;

Создание индикатора на основе модулей торговых сигналов эксперта

В качестве примера создадим индикатор на основе двух модулей сигналов SignalMA и SignalMACD, файлы которых находятся в каталоге MQL5\Include\Expert\Signal.



В мастере MQL5 создадим основу индикатора, выбрав вариант Пользовательский индикатор.

```mql5
//+------------------------------------------------------------------+
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#property indicator_chart_window
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---

//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

Теперь нам нужно указать свойства индикатора, а также определить функции OnInit и OnCalculate.

При создании индикатора на основе модулей торговых сигналов эксперта будем опираться на код сгенерированного с помощью MQL5 эксперта.

```mql5
//+------------------------------------------------------------------+
//| "Tick" event handler function                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   ExtExpert.OnTick();
  }
//+------------------------------------------------------------------+
//| "Trade" event handler function                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
   ExtExpert.OnTrade();
  }
```

При работе такого эксперта, в функции OnTick, вызывается функция OnTick класса CExpert.

```
//+------------------------------------------------------------------+
//| OnTick handler                                                   |
//+------------------------------------------------------------------+
void CExpert::OnTick(void)
  {
//--- check process flag
   if(!m_on_tick_process)
      return;
//--- updated quotes and indicators
   if(!Refresh())
      return;
//--- expert processing
   Processing();
  }
```

В этой функции сначала вызывается функция Refresh, обновляющая цены символа и значения индикаторов, а уже потом вызывается функция Processing, которая получает торговые сигналы и выставляет ордера.

Так как функция Refresh является защищенной, а нам нужно обновлять данные модулей торговых сигналов в нашей функции OnCalculate, создадим свой класс CExpertInd, расширяющий класс CExpert.

```
#include <Expert\Expert.mqh>
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
class CExpertInd : public CExpert
  {
  public:
  virtual bool      RefreshInd(void);
  };
//+------------------------------------------------------------------+
//| Refreshing data for processing                                   |
//+------------------------------------------------------------------+
bool CExpertInd::RefreshInd(void)
  {
   MqlDateTime time;
//--- refresh rates
   if(!m_symbol.RefreshRates())
      return(false);
//--- check need processing
   TimeToStruct(m_symbol.Time(),time);
   if(m_period_flags!=WRONG_VALUE && m_period_flags!=0)
      if((m_period_flags&TimeframesFlags(time))==0)
         return(false);
   m_last_tick_time=time;
//--- refresh indicators
   m_indicators.Refresh();
//--- ok
   return(true);
  }
```

Здесь мы просто перенесли код функции Refresh из класса CExpert, сделав функцию Refresh публичной.

Посмотрев на классы CSignalMA и CSignalMACD, мы увидим, что функции LongCondition и ShortCondition, дающие рыночные модели, вызываются на текущем баре.

Нам же нужно вызывать эти функции на всей истории символа.

Кроме того, нам нужна функция BarsCalculated, возвращающая количество рассчитанных значений в модуле сигналов.

Поэтому создадим классы CSignalMAInd и CSignalMACDInd, расширяющие классы CSignalMA и CSignalMACD.

```
#include <Expert\Signal\SignalMA.mqh>
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
class CSignalMAInd : public CSignalMA
  {
  public:
  virtual int       BarsCalculatedInd();
  virtual int       LongConditionInd(int ind);
  virtual int       ShortConditionInd(int ind);
  );
                                                  |
//+------------------------------------------------------------------+
int CSignalMAInd:: BarsCalculatedInd(){
return m_ma.BarsCalculated();
)

//+------------------------------------------------------------------+
//| "Voting" that price will grow.                                   |
//+------------------------------------------------------------------+
int CSignalMAInd::LongConditionInd(int idx)
  {

   int result=0;

//--- analyze positional relationship of the close price and the indicator at the first analyzed bar
   if(DiffCloseMA(idx)<0.0)
     {
```

Здесь мы определили функцию BarsCalculated, возвращающую количество рассчитанных значений в модуле сигналов.

И изменили функции LongCondition и ShortCondition, передавая в них в качестве параметра индекс бара, на котором нужно вычислить сигнал.

```mql5
#include <Expert\Signal\SignalMA.mqh>
#include <Expert\Signal\SignalMACD.mqh>
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
class CSignalMACDInd : public CSignalMACD
  {
  public:
  virtual int       BarsCalculatedInd();
  virtual int       LongConditionInd(int ind);
  virtual int       ShortConditionInd(int ind);

  };
//+------------------------------------------------------------------+
int CSignalMACDInd:: BarsCalculatedInd(){
return m_MACD.BarsCalculated();
}

//+------------------------------------------------------------------+
//| "Voting" that price will grow.                                   |
//+------------------------------------------------------------------+
int CSignalMACDInd::LongConditionInd(int idx)
  {
   int result=0;

//--- check direction of the main line
   if(DiffMain(idx)>0.0)
     {
```

Тоже самое сделаем и в классе CSignalMACDInd, расширяющем класс CSignalMACD.

Определим функцию BarsCalculated, возвращающую количество рассчитанных значений в модуле сигналов.

И изменим функции LongCondition и ShortCondition, передавая в них в качестве параметра индекс бара, на котором нужно вычислить сигнал.

Теперь можно приступить к коду нашего индикатора.

```mql5
#property indicator_chart_window

#include <Expert\ExpertInd.mqh>
#include <Expert\Signal\SignalMACDInd.mqh>
#include <Expert\Signal\SignalMAInd.mqh>

#property indicator_buffers 2
#property indicator_plots   1
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrBlack,clrRed,clrLawnGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
double        InBuffer[];
double        ColorBuffer[];
int    bars_calculated=0;

input int               Signal_ThresholdOpen        =20;          // Signal threshold value to open [0...100]
input int               Signal_ThresholdClose       =20;          // Signal threshold value to close [0...100]

input int               Signal_MACD_PeriodFast      =12;          // MACD(12,24,9,PRICE_CLOSE) Period of fast EMA
input int               Signal_MACD_PeriodSlow      =24;          // MACD(12,24,9,PRICE_CLOSE) Period of slow EMA
input int               Signal_MACD_PeriodSignal    =9;           // MACD(12,24,9,PRICE_CLOSE) Period of averaging
input ENUM_APPLIED_PRICE Signal_MACD_Applied        =PRICE_CLOSE; // MACD(12,24,9,PRICE_CLOSE) Prices series
input double            Signal_MACD_Weight          =1.0;         // MACD(12,24,9,PRICE_CLOSE) Weight [0...1.0]
input int               Signal_MA_PeriodMA          =12;          // Moving Average(12,0,...) Period of averaging
input int               Signal_MA_Shift             =0;           // Moving Average(12,0,...) Time shift
input ENUM_MA_METHOD    Signal_MA_Method            =MODE_SMA;    // Moving Average(12,0,...) Method of averaging
input ENUM_APPLIED_PRICE Signal_MA_Applied           =PRICE_CLOSE; // Moving Average(12,0,...) Prices series
input double            Signal_MA_Weight            =1.0;         // Moving Average(12,0,...) Weight [0...1.0]

CExpertInd ExtExpert;
CSignalMAInd *filter0 = new CSignalMAInd;
CSignalMACDInd *filter1 = new CSignalMACDInd;
```

Будем рисовать индикатор в виде линии по ценам открытия, которая будет менять цвет в зависимости от прогноза на рост или снижение цены.

И здесь мы взяли входные параметры и код инициализации из кода сгенерированного эксперта.

```
filter0.PeriodMA(Signal_MA_PeriodMA);
filter0.Shift(Signal_MA_Shift);
filter0.Method(Signal_MA_Method);
filter0.Applied(Signal_MA_Applied);

filter1.PeriodFast(Signal_MACD_PeriodFast);
filter1.PeriodSlow(Signal_MACD_PeriodSlow);
filter1.PeriodSignal(Signal_MACD_PeriodSignal);
filter1.Applied(Signal_MACD_Applied);

signal.AddFilter(filter0);
signal.AddFilter(filter1);
if(!ExtExpert.ValidationSettings())
     {
      //--- failed
      ExtExpert.Deinit();
      return(INIT_FAILED);
     }
//--- Tuning of all necessary indicators
   if(!ExtExpert.InitIndicators())
     {
      //--- failed
      printf(__FUNCTION__+": error initializing indicators");
      ExtExpert.Deinit();
      return(INIT_FAILED);
     }
//--- ok
   //--- indicator buffers mapping
   SetIndexBuffer(0,InBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorBuffer,INDICATOR_COLOR_INDEX);

ArraySetAsSeries(InBuffer,true);
ArraySetAsSeries(ColorBuffer,true);
```

Так как модули сигналов работают с данными как с таймсериями, применим функцию ArraySetAsSeries к буферам нашего индикатора.

```
 bars_calculated=calculated;

 ArraySetAsSeries(open,true);

 ExtExpert.RefreshInd();

 if(values_to_copy>1)
{

for (int i=0; i<values_to_copy; i++){

ColorBuffer[i]=0;
InBuffer[i]=open[i];

double result0=Signal_MA_Weight*(filter0.LongConditionInd(i)-filter0.ShortConditionInd(i));
double result1=Signal_MACD_Weight*(filter1.LongConditionInd(i)-filter1.ShortConditionInd(i));
double result=(result0+result1)/2;

if(result>=Signal_ThresholdOpen)
     {
 ColorBuffer[i]=2;
     }

 if(-result>=Signal_ThresholdOpen){
 ColorBuffer[i]=1;
 }
}
}
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

В функции OnCalculate индикатора мы сначала обновляем все данные, а затем получаем взвешенные сигналы модулей и сравниваем их с пороговым значением.

В итоге получаем прогноз на увеличение или уменьшение цены.

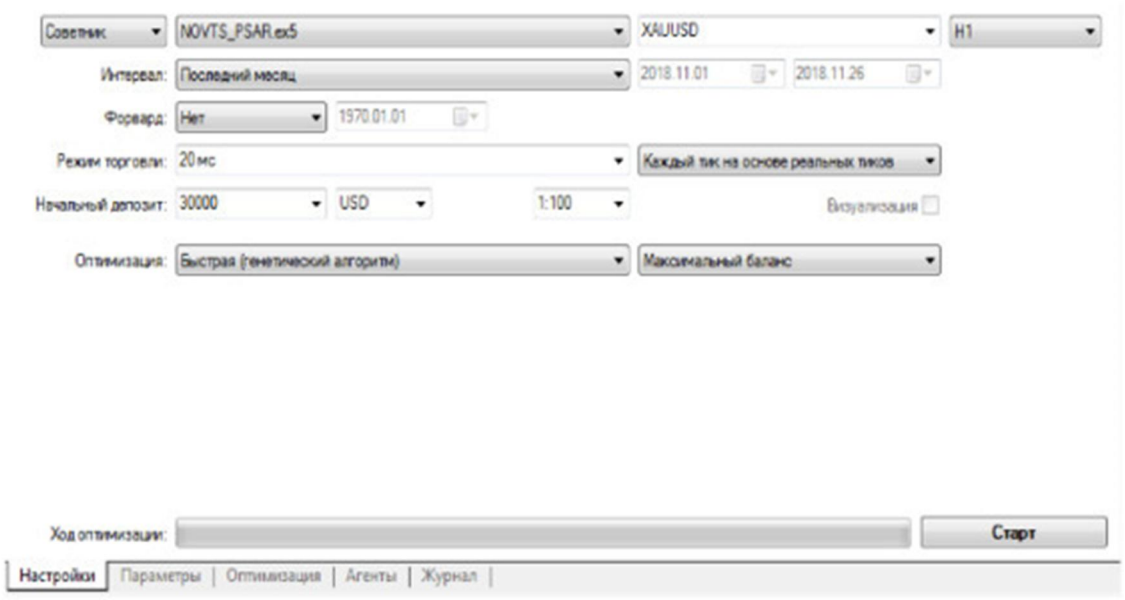


После присоединения к графику нашего индикатора на основе модулей сигналов советника, мы увидим, как индикатор меняет цвет в зависимости от прогноза на рост или снижение цены.

Создание такого индикатора – это способ наглядно посмотреть работу советника на графике символа.

Генетические алгоритмы

Если рассматривать задачу создания самооптимизирующегося советника,



То при создании самооптимизирующегося советника, на определенном этапе его работы, требуется автоматический вызов кода, который заново оптимизирует параметры советника на истории финансового инструмента, и далее советник продолжит свою работу уже с новыми параметрами.

Так как сам советник работает на текущем баре и не использует историю символа, код оптимизации параметров советника должен опираться на код индикатора, который в свою очередь создан на основе кода советника.

Для оптимизации параметров советника, или теперь уже параметров индикатора, совпадающих с параметрами советника, можно использовать полный перебор, однако для сокращения времени оптимизации можно применить генетический алгоритм.

Для начала обратимся к терминологии.

Хромосома состоит из набора генов – набора случайно выбранных параметров советника в допустимых диапазонах.

Поколение это набор хромосом.

Фитнес функция FF это код, возвращающий значение советника, по которому производится оптимизация

$$p(x_i^t) = \frac{f(x_i^t)}{\sum_{j=1}^{\lambda} f(x_j^t)}$$

$f(x_i)$ - значение функции фитнеса VFF

Хромосома состоит из набора генов – набора случайно выбранных параметров советника в допустимых диапазонах.

Поколение это набор хромосом.

Фитнес функция FF это код, возвращающий значение советника, по которому производится оптимизация, например прибыль.

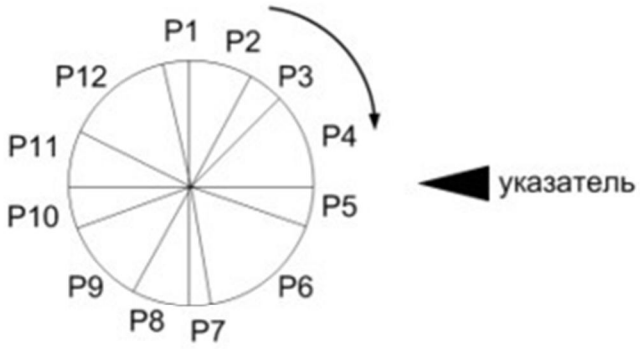
Значение функции фитнеса VFF используется для вычисления вероятности P воспроизведения хромосомы – члена популяции.

Начальная популяция формируется случайным образом и размер популяции (количество особей) фиксируется и не изменяется в течение работы всего алгоритма.

На основе вероятностей воспроизведения хромосом начальной популяции формируется новая популяция, и так далее пока не сработает условие завершения работы алгоритма.

Каждая следующая популяция формируется из предыдущей с помощью отбора, скрещивания и мутации.

Для отбора могут использоваться такие алгоритмы как рулетка, турнирный отбор и элитный отбор.



Колесо рулетки

$$M = P(x_i) * N$$

где N – число особей в популяции

При применении рулетки, колесо рулетки делится на сектора, число которых совпадает с числом особей популяции.

Площадь каждого сектора пропорциональна вероятности воспроизведения особи популяции.

Отбор производится с помощью вращения колеса рулетки.

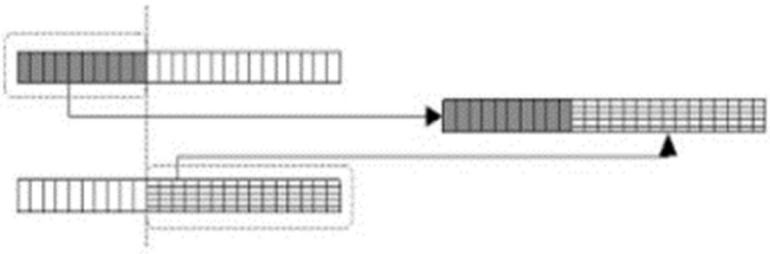
Таким образом, особь с наибольшим значением вероятности воспроизведения попадает в следующую популяцию наибольшее число раз.

Количество копий особи в следующей популяции определяется по формуле, как показано на слайде, где N – число особей в популяции.



При турнирном отборе из популяции случайно выбираются две особи, из которых остается особь с наибольшим значением вероятности воспроизведения.

При элитном отборе несколько лучших особей переходят в следующую популяцию без изменений, не участвуя в отборе и скрещивании.



После отбора идет скрещивание особей, при котором сначала выбираются две особи, затем случайно определяется точка разрыва

в диапазоне числа параметров фитнес функции, после чего особи обмениваются сегментами хромосомы.

Само скрещивание производится с вероятностью ½.

При применении мутации, сначала случайно выбирается параметр фитнес функции, затем он модифицируется.

Сама мутация производится с вероятностью 0,001.

```
//+------------------------------------------------------------------+
//Библиотека "Универсального Генетического Алгоритма UGAlib"        |
//использующего представление хромосомы вещественными числами.      |
//+------------------------------------------------------------------+

//---------------------Глобальные переменные-----------------------------
double Chromosome[];             //Набор оптимизируемых аргументов функции - генов
                                 //(например: веса нейронной сети и т.д.)-хромосома
int    ChromosomeCount     =0;  //Максимально возможное количество хромосом в колонии
int    TotalOfChromosomesInHistory=0;//Общее количество хромосом в истории
int    ChrCountInHistory   =0;  //Количество уникальных хромосом в базе хромосом
int    GeneCount           =0;  //Количество генов в хромосоме

double RangeMinimum        =0.0;//Минимум диапазона поиска
double RangeMaximum        =0.0;//Максимум диапазона поиска
double Precision           =0.0;//Шаг поиска
int    OptimizeMethod      =0;  //1-минимум, любое другое - максимум

double Population   [][1000];   //Популяция
double Colony       [][500];    //Колония потомков
int    PopulChromosCount   =0;  //Текущее количество хромосом в популяции
int    Epoch               =0;  //Кол-во эпох без улучшения
int    AmountStartsFF=0;        //Количество запусков функции приспособленности
```

Библиотека UGAlib MQL5 Community реализации генетического алгоритма использует представление хромосомы в виде массива Chromosome, где 0 индекс это значение фитнес функции, а остальные индексы это параметры фитнес функции или гены хромосомы.

Оптимизация параметров фитнес функции ведется в одном диапазоне от RangeMinimum до RangeMaximum.

Поэтому если оптимизируемые параметры имеют разные диапазоны, их нужно привести к одному диапазону, как правило, от 0.0 до 1.0.

Популяция особей представлена двумерным массивом Population, где строки это хромосомы.

Популяция ранжируется по значению фитнес функции таким образом, что первый ее член имеет наилучшее значение фитнес функции по критерию оптимизации.

Популяция делится на две части.

```
//=============================================================================
// 1) Создать протопопуляцию                                          ------1)
ProtopopulationBuilding ();
//=============================================================================
// 2) Определить приспособленность каждой особи                       ------2)
//Для 1-ой колонии
for (chromos=0;chromos<ChromosomeCount;chromos++)
  for (gene=1;gene<=GeneCount;gene++)
    Colony[gene][chromos]=Population[gene][chromos];

GetFitness(historyHromosomes);

for (chromos=0;chromos<ChromosomeCount;chromos++)
  Population[0][chromos]=Colony[0][chromos];

//Для 2-ой колонии
for (chromos=ChromosomeCount;chromos<ChromosomeCount*2;chromos++)
  for (gene=1;gene<=GeneCount;gene++)
    Colony[gene][chromos-ChromosomeCount]=Population[gene][chromos];

GetFitness(historyHromosomes);

for (chromos=ChromosomeCount;chromos<ChromosomeCount*2;chromos++)
  Population[0][chromos]=Colony[0][chromos-ChromosomeCount];
```

Вначале вся популяция заселяется особями со случайно выбранными генами в диапазоне.

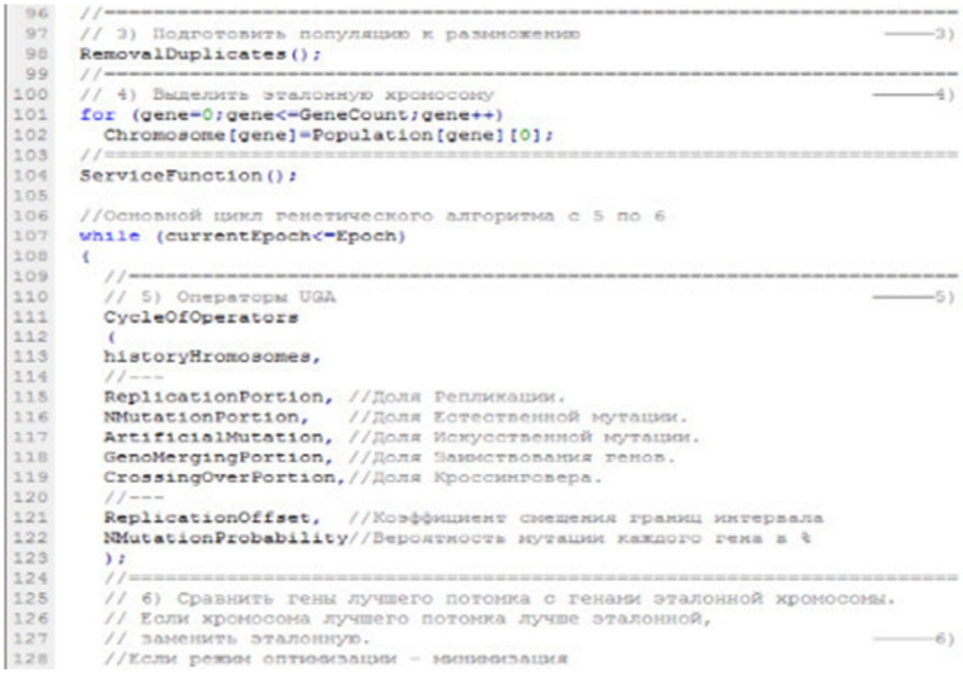
Затем для этих особей вычисляется значение фитнес функции, которое помещается в 0 индекс хромосомы.

При этом функция GetFitness расчета значения фитнес функции оперирует колонией потомков, представленной массивом Colony, имеющим размер в два раза меньший, чем размер популяции.

Таким образом, популяция заселяется двумя колониями потомков.

Колония потомков имеет размер меньший, чем размер популяции, для того, чтобы после мутаций и скрещиваний заселить ту часть популяции, которая имеет худшие значения фитнес функции.

При этом особи, полученные в результате мутаций и скрещиваний, заселяют именно колонию потомков.



```
//=============================================================================
// 3) Подготовить популяцию к размножению                              ———3)
RemovalDuplicates();
//=============================================================================
// 4) Выделить эталонную хромосому                                     ———4)
for (gene=0;gene<=GeneCount;gene++)
  Chromosome[gene]=Population[gene][0];
//=============================================================================
ServiceFunction();

//Основной цикл генетического алгоритма с 5 по 6
while (currentEpoch<=Epoch)
{
  //=============================================================================
  // 5) Операторы UGA                                                  ———5)
  CycleOfOperators
  (
  historyHromosomes,
  //---
  ReplicationPortion, //Доля Репликации.
  NMutationPortion,   //Доля Естественной мутации.
  ArtificialMutation, //Доля Искусственной мутации.
  GenoMergingPortion, //Доля Заимствования генов.
  CrossingOverPortion,//Доля Кроссинговера.
  //---
  ReplicationOffset,  //Коэффициент смещения границ интервала
  NMutationProbability//Вероятность мутации каждого гена в %
  );
  //=============================================================================
  // 6) Сравнить гены лучшего потомка с генами эталонной хромосомы.
  // Если хромосома лучшего потомка лучше эталонной,
  // заменить эталонную.                                               ———6)
  //Если режим оптимизации - минимизация
```

После начального заселения популяции производится удаление дубликатов с помощью функции RemovalDuplicates, в которой также производится ранжирование популяции по значению фитнес функции.

После подготовки начальной популяции вызывается цикл эпох – цикл рождения новых популяций, который продолжается до тех пор, пока количество эпох без улучшения не превысит порог Epoch.

Критерием улучшения служит эталонная хромосома – первая особь популяции.

В цикле эпох популяция модифицируется с помощью репликации, естественной мутации, искусственной мутации, заимствования генов и скрещивания, применяемых для заселения новой колонии потомков, которая затем замещает часть популяции, имеющей худшие значения фитнес функции.

В цикле заселения новой колонии потомков функции CycleOfOperators, операторы репликации, естественной мутации, искусственной мутации, заимствования генов и скрещивания выбираются случайно, в зависимости от их доли от 0 до 100.

После создания новой популяции, в ней также удаляются дубликаты, и она ранжируется по значению фитнес функции.

Здесь репликация заключается в выборе двух хромосом популяции и создании на их основе новой хромосомы, для которой гены случайно выбираются в расширенном диапазоне с помощью сдвига от гена первой особи влево и от гена второй особи вправо.

Выбор двух родителей из популяции осуществляется с помощью алгоритма рулетки, упомянутого выше.

Естественная мутация производится с помощью выбора одного родителя из популяции, используя алгоритм рулетки, и замены его генов генами, случайно выбранными в диапазоне от RangeMinimum до RangeMaximum.

Замена генов производится с вероятностью NMutationProbability от 0.0 до 100.0.

При искусственной мутации выбираются два родителя из популяции, используя алгоритм рулетки, и гены потомка случайно выбираются из незанятого генами родителей пространства на числовой прямой в диапазонах от RangeMinimum до сдвига от гена первой особи вправо и от сдвига от гена второй особи влево до RangeMaximum.

При заимствовании генов для первого гена потомка выбирается родитель из популяции, используя алгоритм рулетки, и берется у него первый ген, далее, для второго гена отбирается второй родитель и берется второй ген и т. д.

При скрещивании выбираются два родителя из популяции, используя алгоритм рулетки, и гены потомка формируются за счет обмена отрезками хромосом мамы и папы.

Полный код реализации генетического алгоритма можно посмотреть в файле UGAlib.

Для использования библиотеки UGAlib необходимо написать две функции FitnessFunction и ServiceFunction.

Функция FitnessFunction получает на вход индекс хромосомы и рассчитывает для нее значение, по которому ведется оптимизация генов хромосомы.

Функция ServiceFunction может выводить значение фитнес функции и остальные гены эталонной хромосомы при каждом проходе оптимизации.

В качестве примера рассмотрим оптимизацию параметров индикатора, созданного в предыдущей лекции.

```
if(result>=Signal_ThresholdOpen)
     {
ColorBuffer[i]=2;
if(flagSell==true){
flagSell=false;
priceStopSell=InBuffer[i];
profitSell=(priceStopSell-priceSell-sp)*10000;
if(profitSell>0){countProfitSellPlus++;}else{countProfitSellMinus++;}
profitTotalSell=profitTotalSell+profitSell;
 }
 if(flagBuy==false){
 flagBuy=true;
 priceBuy=InBuffer[i];
 }
     }

if(-result>=Signal_ThresholdOpen){
ColorBuffer[i]=1;
if(flagBuy==true){
flagBuy=false;
priceStopBuy=InBuffer[i];
profitBuy=(priceStopBuy-priceBuy-sp)*10000;
if(profitBuy>0){countProfitBuyPlus++;}else{countProfitBuyMinus++;}
profitTotalBuy=profitTotalBuy+profitBuy;
 }
 if(flagSell==false){
priceSell=InBuffer[i];
flagSell=true;
 }
 }
}
//Print(" ProfitBuy ", profitTotalBuy," countProfitBuyPlus ",countProfitBuy
//Print(" ProfitSell ", profitTotalSell," countProfitSellPlus ",countProfit
```

Модифицируем код индикатора таким образом, чтобы рассчитывать виртуальные сделки на покупку и продажу финансового инструмента по сигналам индикатора.

Здесь мы добавили вычисление виртуального профита и счетчик выигрышных сделок.

Напишем фитнес функцию на основе этого индикатора.

```
void FitnessFunction(int chromos)
{

double _MACD_Weight=0.0;
double _MA_Weight=0.0;
double sum=0.0;
int    cnt=1;

while (cnt<=GeneCount)
  {
    _MACD_Weight=Colony[cnt][chromos];
    cnt++;
    _MA_Weight=Colony[cnt][chromos];
    cnt++;
```

Оптимизировать будем весы сигналов MA и MACD для получения максимального профита.

```
//+------------------------------------------------------------------+
//|                                                    GenScript.mq5 |
//|                                          Copyright 2018, NOVTS |
//|                                                 http://novts.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2018, NOVTS"
#property link      "http://novts.com"
#property version   "1.00"
#include <MAMACDFitness.mqh>
#include <UGAlib.mqh>

double ReplicationPortion_E  = 100.0; //Доля Репликации.
double NMutationPortion_E    = 10.0;  //Доля Естественной мутации.
double ArtificialMutation_E  = 10.0;  //Доля Искусственной мутации.
double GenoMergingPortion_E  = 20.0;  //Доля Заимствования генов.
double CrossingOverPortion_E = 20.0;  //Доля Кроссинговера.
//---
double ReplicationOffset_E   = 0.5;   //Коэффициент смещения границ интервала
double NMutationProbability_E= 5.0;   //Вероятность мутации каждого гена в %
```

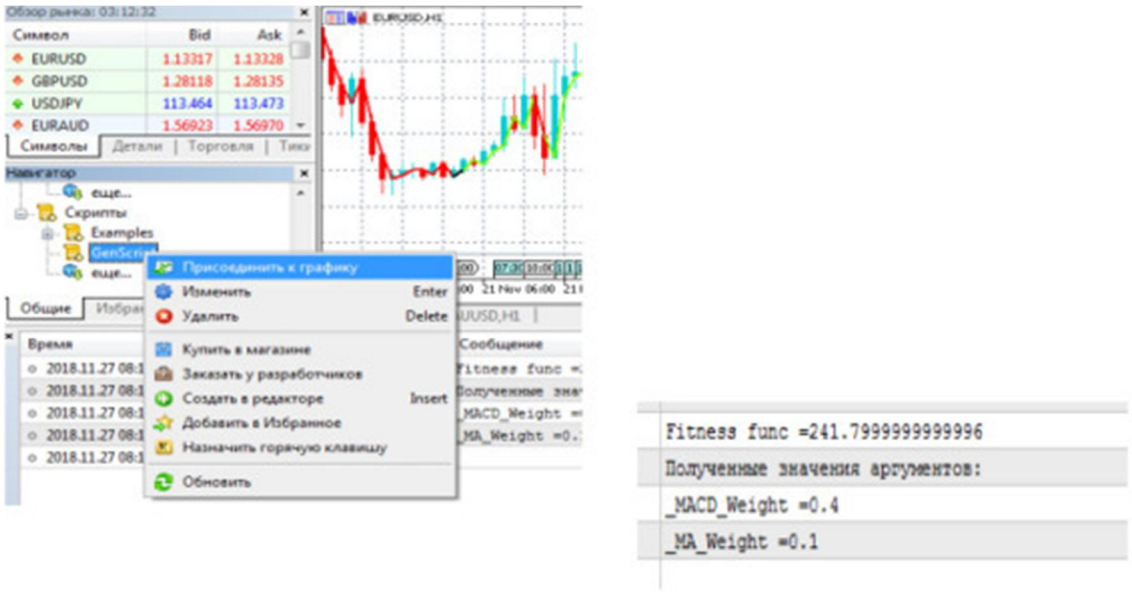
И напишем скрипт, который будет оптимизировать параметры индикатора с помощью генетического алгоритма.

```
void OnStart()
  {
//---
ChromosomeCount      = 10;     //Кол-во хромосом в колонии
GeneCount            = 2;      //Кол-во генов
Epoch                = 50;     //Кол-во эпох без улучшения
//---
RangeMinimum         = 0.0;  //Минимум диапазона поиска
RangeMaximum         = 1.0;    //Максимум диапазона поиска
Precision            = 0.1;//Требуемая точность
OptimizeMethod       = 2;      //Оптим.:1-Min,другое-Max
ArrayResize(Chromosome,GeneCount+1);
ArrayInitialize(Chromosome,0);
//Локальные переменные
  int time_start=(int)GetTickCount(),time_end=0;
//Запуск главной ф-ии UGA
  UGA
  (
  ReplicationPortion_E, //Доля Репликации.
  NMutationPortion_E,   //Доля Естественной мутации.
  ArtificialMutation_E, //Доля Искусственной мутации.
  GenoMergingPortion_E, //Доля Заимствования генов.
  CrossingOverPortion_E,//Доля Кроссинговера.
  //---
  ReplicationOffset_E,  //Коэффициент смещения границ интервала
  NMutationProbability_E//Вероятность мутации каждого гена в %
  );
  //-----------------------------------
  time_end=(int)GetTickCount();
  //-----------------------------------
  Print(time_end-time_start," мс - Время исполнения");
  //-----------------------------------

  }
```

Здесь в функции OnStart скрипта мы вызываем главную функцию UGA генетического алгоритма, которая будет использовать написанные нами

функции FitnessFunction и ServiceFunction во включаемом файле MAMACDFitness.



Запустив скрипт, путем присоединения его к графику символа, мы получим следующий результат, который даст нам оптимальные веса сигналов.

Используемый индикатор основан на применении классов CiMA и CiMACD, имеющих проблемы с глубиной истории, потому параметр size в фитнес функции не может быть большим.

Поэтому перепишем индикатор, используя более низкоуровневый интерфейс.

```mql5
#include <Expert\Signal\SignalMA.mqh>
class CSignalMAIndLow : public CSignalMA
  {
  public:
  virtual int       BarsCalculatedInd();
  virtual int       LongConditionInd(int ind, int amount, double close, double open, double low);
  virtual int       ShortConditionInd(int ind, int amount, double close, double open, double high);
  };
```

```mql5
//+------------------------------------------------------------------+
#include <Expert\Signal\SignalMACD.mqh>

class CSignalMACDIndLow : public CSignalMACD
  {
  public:
  virtual int       BarsCalculatedInd();
  virtual int       LongConditionInd(int ind, int amount, double &low[], double &high[]);
  virtual int       ShortConditionInd(int ind, int amount, double &low[], double &high[]);
  protected:
  int               StateMain(int ind,double &Main[]);
  bool              ExtState(int ind, double &Main[], double &low[], double &high[]);
  };
```

Создадим классы сигналов CSignalMACDIndLow и CSignalMAIndLow.

```mql5
int CSignalMAIndLow::LongConditionInd(int idx, int amount, double close, double open, double low)
  {
  int handle=m_ma.Handle();
  double         iMABuffer[];
   if(CopyBuffer(handle,0,0,amount,iMABuffer)<0)
     {
      //--- если копирование не удалось, сообщим код ошибки
      PrintFormat("Не удалось скопировать данные из индикатора iMA, код ошибки %d",GetLastError());
      //--- завершим с нулевым результатом - это означает, что индикатор будет считаться нерассчитанным
      return(-1);
     }
   ArraySetAsSeries(iMABuffer,true);

   int result=0;

   double DiffCloseMA = close - iMABuffer[idx];
   double DiffOpenMA = open - iMABuffer[idx];
   double DiffMA = iMABuffer[idx] - iMABuffer[idx+1];
   double DiffLowMA = low - iMABuffer[idx];
```

В которых в методах LongCondition и ShortCondition мы будем использовать не классы CiMA и CiMACD, а хэндлы индикаторов.

```
for (int i=0; i<(values_to_copy-2); i++){

ColorBuffer[i]=0;
InBuffer[i]=open[i];

double result0=Signal_MA_Weight*(filter0.LongConditionInd(i, values_to_copy, close[i], open[i],
low[i])-filter0.ShortConditionInd(i, values_to_copy, close[i], open[i], high[i]));

double result1=Signal_MACD_Weight*(filter1.LongConditionInd(i, values_to_copy,
_low, _high)-filter1.ShortConditionInd(i, values_to_copy, _low, _high));

double result=(result0+result1)/2;
```

Соответственно в индикаторе будем вызывать эти переписанные функции.

```
int handleInd;
double BufferInd[];
double BufferColorInd[];

handleInd=iCustom(NULL,0,"IndSignalsLow",
Signal_ThresholdOpen,
Signal_ThresholdClose,
Signal_MACD_PeriodFast,
Signal_MACD_PeriodSlow,
Signal_MACD_PeriodSignal,
Signal_MACD_Applied,
_MACD_Weight,
Signal_MA_PeriodMA,
Signal_MA_Shift,
Signal_MA_Method,
Signal_MA_Applied,
_MA_Weight
);
```

Включим этот переписанный индикатор в фитнес функцию.

И запустим скрипт.

Глубина истории увеличится, но при этом существенно увеличится и время оптимизации – на порядок, при том же самом результате на 1000 барах.

Поэтому для самооптимизации советника будем использовать первый вариант индикатора с глубиной истории в 1000 бар.

С помощью мастера MQL5 сгенерируем код советника на основе сигналов MA и MACD и добавим в него самооптимизацию параметров Signal_MA_Weight и Signal_MACD_Weight с использованием приведенной выше фитнес функции.

```
void OnTick()
  {
  if(AccountInfoDouble(ACCOUNT_BALANCE)>balance) balance=AccountInfoDouble(ACCOUNT_BALANCE);
   double bd = ((balance-AccountInfoDouble(ACCOUNT_BALANCE))/balance)*100;
   if(bd>10){
   UGA
  (
  ReplicationPortion_E, //Доля Репликации.
  NMutationPortion_E,   //Доля Естественной мутации.
  ArtificialMutation_E, //Доля Искусственной мутации.
  GenoMergingPortion_E, //Доля Заимствования генов.
  CrossingOverPortion_E,//Доля Кроссинговера.
  //---
  ReplicationOffset_E,  //Коэффициент смещения границ интервала
  NMutationProbability_E//Вероятность мутации каждого гена в %
  );
double _MACD_Weight=0.0;
double _MA_Weight=0.0;
int    cnt=1;
    while (cnt<=GeneCount)
    {
      _MACD_Weight=Chromosome[cnt];
      cnt++;
      _MA_Weight=Chromosome[cnt];
      cnt++;
    }
filter0.Weight(_MA_Weight);
filter1.Weight(_MACD_Weight);

balance=AccountInfoDouble(ACCOUNT_BALANCE);
```

Здесь в функции OnTick при превышении порога просадки баланса вызывается функция UGA генетического алгоритма, которая оптимизирует веса сигналов.

И в фитнес функцию перед копированием буферов индикатора добавим вызов Sleep для того, чтобы индикатор успел рассчитаться.

При тестировании советника на паре EURUSD на часах без самооптимизации получаем следующий результат.

Чистая прибыль: 26.00	
Абсолютная просадка по балансу:	4 431.00
Абсолютная просадка по средствам:	4 550.00
Общая прибыль: 79 532.00	
Максимальная просадка по балансу:	5 862.00 (50.08%)
Максимальная просадка по средствам:	5 956.00 (50.57%)
Общий убыток: -79 506.00	
Относительная просадка по балансу:	50.08% (5 862.00)
Относительная просадка по средствам:	50.57% (5 956.00)
Прибыльность: 1.00	
Матожидание выигрыша: 0.04	

С включенной самооптимизацией советника при тестировании получаем следующий результат.

Чистая прибыль: 384.00	
Абсолютная просадка по балансу:	504.00
Абсолютная просадка по средствам:	612.00
Общая прибыль: 4 004.00	
Максимальная просадка по балансу:	1 276.00 (10.94%)
Максимальная просадка по средствам:	1 341.00 (11.44%)
Общий убыток: -3 620.00	
Относительная просадка по балансу:	10.94% (1 276.00)
Относительная просадка по средствам:	12.43% (1 333.00)
Прибыльность: 1.11	
Матожидание выигрыша: 24.00	

Как мы можем видеть, матожидание прибыли существенно выросло.

Использование библиотечных классов и функций

В функции OnTick советника, для того чтобы торговать только при появлении нового бара на графике символа, мы использовали структуру данных datetime, локальную статическую переменную и функцию CopyTime, для того чтобы отслеживать время открытия бара.

```
static datetime Old_Time;
datetime New_Time[1];
bool IsNewBar=false;

int copied=CopyTime(_Symbol,PERIOD_CURRENT,0,1,New_Time);
if(copied>0)
  {
   if(Old_Time!=New_Time[0])
     {
      IsNewBar=true;
      Old_Time=New_Time[0];
     }
  }
else
  {
   Alert("Ошибка копирования времени, номер ошибки =",GetLastError());
   ResetLastError();
   return;
  }

if(IsNewBar==false)
  {
  return;
  }
```

Тоже самое можно сделать с помощью функции iTime.

```
if (!IsNewCandle())
return;


bool IsNewCandle(void)
{
  static datetime prevTime = 0;
  datetime currTime = iTime(_Symbol, PERIOD_CURRENT, 0);

  if (prevTime != currTime)
  {
    prevTime = currTime;
    return (true);
  }
  return (false);
}
```

Здесь мы также используем структуру данных datetime и локальную статическую переменную.

Но вместо функции CopyTime, мы используем функцию iTime для получения времени открытия бара.

Теперь, для открытия позиции, мы использовали структуру MqlTradeRequest и функцию OrderSend.

```
      mrequest.action = TRADE_ACTION_DEAL;
      mrequest.price = NormalizeDouble(latest_price.ask,_Digits);
      mrequest.symbol = _Symbol;
      mrequest.volume = Lot;
      mrequest.magic = EA_Magic;
      mrequest.type = ORDER_TYPE_BUY;
      mrequest.type_filling = ORDER_FILLING_FOK;
      mrequest.deviation=100;
      mrequest.sl = NormalizeDouble(latest_price.bid - SLB,_Digits);
      mrequest.tp = NormalizeDouble(latest_price.ask + ProfitBuy,_Digits);

     if(!OrderCheck(mrequest,check_result))
  {
   return;
  }else{
      if(OrderSend(mrequest,mresult)){}
}
      if(mresult.retcode==10009 || mresult.retcode==10008)
        {
        priceBuy=mresult.price;
        slBuy= mrequest.sl;
        }
      else
        {
         return;
        }
```

Тоже самое можно сделать с помощью классов CTrade и CPositionInfo.

```
CTrade trd;
CPositionInfo pos;

void OpenBuy(double volume)
{
  if (trd.Buy(volume, _Symbol, Ask(), 0, 0, ""))
  {
    ulong ticket = trd.ResultDeal();
    pos.SelectByTicket(ticket);

    double sl = 0;
    double tp = 0;

    if (inpStopLoss > 0)
      sl = ND(pos.PriceOpen() - pos.PriceOpen() * (inpStopLoss / 100));

    if (inpTakeProfit > 0)
      tp = ND(pos.PriceOpen() + pos.PriceOpen() * (inpTakeProfit / 100));

    if (sl == 0 && tp == 0)
      return;

    trd.PositionModify(ticket, sl, tp);
  }
}
```

Здесь мы с помощью метода Buy класса CTrade открываем длинную позицию с указанным объемом, на текущем символе, по цене, которая возвращается функцией Ask, с нулевым стоплоссом и тейкпрофитом.

Затем мы получаем тикет сделки с помощью метода ResultDeal, и фиксируем позицию CPositionInfo с помощью метода SelectByTicket.

Затем мы вычисляем стоплосс и тейкпрофит на основе входных параметров, и модифицируем открытую позицию с помощью метода PositionModify класса CTrade.

```
//+------------------------------------------------------------------+
//| Функция возвращает цену Ask
//+------------------------------------------------------------------+
double Ask(void)
{
   return (SymbolInfoDouble(_Symbol, SYMBOL_ASK));
}
```

```
//+------------------------------------------------------------------+
//| Функция нормализует цену
//+------------------------------------------------------------------+
double ND(double price)
{
   return (NormalizeDouble(price, _Digits));
}
```

Здесь показаны используемые функции, которые возвращают цену предложения на покупку от брокера, и округляют цену до указанной точности.

```
void CheckTrailingStop(void)
{
   if (PositionSelect(_Symbol))
   {
      ulong  ticket = pos.Identifier();
      long   type   = pos.PositionType();
      double op     = pos.PriceOpen();
      double sl     = pos.StopLoss();

      if (type == POSITION_TYPE_BUY)
         if (op + inpTrailingStop * _Point <= Bid())
         {
            if (op > sl || sl == 0)
            {
               sl = Bid() - inpTrailingStopStep * _Point;
               trd.PositionModify(ticket, sl, 0);
            }
            if (sl + inpTrailingStopStep * 2 * _Point <= Bid())
            {
               sl = Bid() - inpTrailingStopStep * _Point;
               trd.PositionModify(ticket, sl, 0);
            }
         }
      if (type == POSITION_TYPE_SELL)
         if (op - inpTrailingStop * _Point >= Ask())
         {
            if (op < sl || sl == 0)
            {
               sl = Ask() + inpTrailingStopStep * _Point;
               trd.PositionModify(ticket, sl, 0);
            }
            if (sl - inpTrailingStopStep * 2 * _Point >= Ask())
            {
               sl = Ask() + inpTrailingStopStep * _Point;
               trd.PositionModify(ticket, sl, 0);
            }
         }
   }
}
```

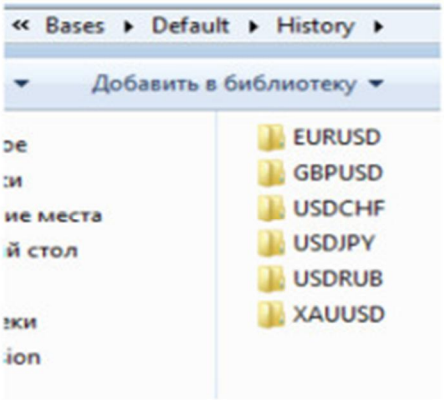
Используя методы классов CTrade и CPositionInfo можно также реализовать трейлинг позиций на покупку и продажу.

И еще раз о тестировании роботов

Любой робот, который вы создадите, будет иметь входные параметры своей работы, которые нужно будет оптимизировать для конкретной валютной пары.

Причем оптимизацию параметров робота придется проводить периодически, подстраивая их под текущее поведение рынка.

Невозможно создать универсального робота, который бы выдавал одинаковые результаты для любого символа и для любого периода времени.



Для оптимизации параметров робота, его нужно протестировать на истории цен, которую терминал загружает с сервера брокера и сохраняет в папке Bases\Default\History.

Данные истории цен закачиваются с торгового сервера по запросу терминала в виде минутных баров.

Однако вот какое дело, если вы установите терминалы от разных брокеров, вы обнаружите, что каждый брокер будет предоставлять свою историю цен.

И один и тот же робот будет иметь разные оптимизированные параметры для разных брокеров.

Универсальных оптимизированных параметров робота вы не получите.

Более того, один и тот же робот с одними и теми же параметрами будет по-разному торговать на терминалах разных брокеров.

Если же сравнить торговлю робота в реальном времени на терминале брокера с последующим тестированием этого же робота на том же промежутке времени, результаты будут совпадать.

Так что брокер со своей историей не химичит.

При чувствительности результатов работы робота к оптимизации параметров робота, возникает вопрос, как же настроить робота, чтобы он обеспечил прибыльную торговлю на реальном рынке?

Тот, кто найдет ответ на этот вопрос, будет сказочно богат.

Один из способов оптимизации – выбрать участок истории, который, как вам кажется, будет похож на движение цены в будущем, и провести оптимизацию робота на этом временном отрезке ценовой истории.

Для автоматизированной торговли с использованием робота, сначала нужно не кодировать, а создать и сформировать для себя философию рынка и торговли на нем.

Сначала нужно разобраться в движущих силах и механизмах рынка.

Только после этого, на основе своей философии, можно выбрать стратегию торговли и реализовать ее в коде советника.

В качестве демонстрации я покажу работу торгового робота, созданного по такой технологии.

Торговля 2 неделя

Чистая прибыль	1 135.28
Общая прибыль	2 543.72
Общий убыток	-1 408.44

Торговля 3 неделя

Чистая прибыль	349.31
Общая прибыль	2 246.56
Общий убыток	-1 897.25

Торговля 4 неделя

Чистая прибыль	158.64
Общая прибыль	1 477.78
Общий убыток	-1 319.14

Любой советник требует периодической настройки своих параметров.

Значения параметров советника по сути отражают настроение на рынке.

И так как настроение на рынке периодически меняется, необходимо перенастраивать советник.

Настройка советника производится с помощью тестера стратегий терминала.

Так как в тестере стратегий могут устанавливаться разные режимы задержек и тиков, необходимо исследовать работу сервера брокера, чтобы понять, какому режиму тестера соответствует работа сервера.

Исследовать сервер брокера можно следующим образом.

Можно запустить торговать свой советник, а затем по прошествии определенного периода времени тестировать советник на этом периоде времени, меняя параметры тестера таким образом, чтобы тестирование максимально совпало с реальной торговлей.

Здесь, в нашем примере, мы настраиваем свой советник каждую неделю на периоде прошедшей недели и запускаем торговать таким образом настроенный советник в течение следующей недели.

Как мы видим, наш советник стабильно выдает прибыль.



Однако в период отсутствия тренда, в период высокой волатильности, как показано на 4-х часовом графике, прибыльный советник будет выдавать убыток.

Признаком приближения такого периода может служить снижение прибыльности советника.

В такие периоды лучше не торговать.